

19980924 064

for Improved Generalization
in Artificial Neural Networks

DISSERTATION

Lemuel Ray Myers, Jr.
Captain, USAF

AFIT/DS/ENG/98-14

DTIC QUALITY INSPECTED 1

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/DS/ENG/98-14

Radial Complexity Estimation
for Improved Generalization
in Artificial Neural Networks

DISSERTATION
Lemuel Ray Myers, Jr.
Captain, USAF

AFIT/DS/ENG/98-14

Approved for public release; distribution unlimited

The views expressed in this dissertation are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

Radial Complexity Estimation
for Improved Generalization
in Artificial Neural Networks

DISSERTATION

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Doctor of Philosophy

Lemuel Ray Myers, Jr., B.S.E.E., M.S.E.E.
Captain, USAF

September, 1998

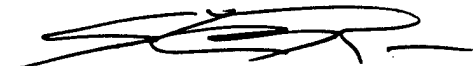

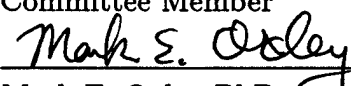
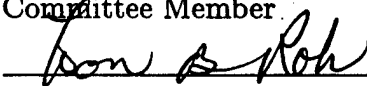
Approved for public release; distribution unlimited

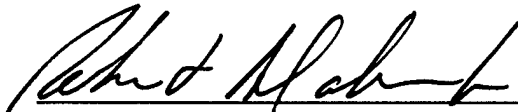
Radial Complexity Estimation
for Improved Generalization
in Artificial Neural Networks

Lemuel Ray Myers, Jr., B.S.E.E., M.S.E.E.

Captain, USAF

Approved:

	<u>4 Sept 1998</u>
Steven K. Rogers PhD Research Advisor	Date
	<u>4 Sept 1998</u>
Matthew Kabrisky PhD Committee Member	Date
	<u>4 Sept 1998</u>
Mark E. Oxley PhD Committee Member	Date
	<u>4 Sep 98</u>
Won B. Roh PhD Dean's Representative	Date


Robert A. Calico, Jr
Dean

Abstract

When training an artificial neural network (ANN) for classification using backpropagation of error, the weights are usually updated by minimizing the sum-squared error on the training set. As training ensues, overtraining may be observed as the network begins to memorize the training data. This occurs because, as the magnitude of the weight vector, $\|\mathbf{w}\|$, grows, the decision boundaries become overly complex in much the same way as a too-high order polynomial approximation can overfit a data set in a regression problem. Since $\|\mathbf{w}\|$ grows during standard backpropagation, it is important to initialize the weights with consideration to the importance of the weight vector magnitude, $\|\mathbf{w}\|$. With this in mind, the expected value of the magnitude of the initial weight vector is here derived for the separate cases of each weight drawn from a normal or uniform distribution. The usefulness of this derivation is universal since the magnitude of the weight vector plays such an important role in the formation of the classification boundaries. When the network overtrains on the training data, it will not exhibit consistently low error on subsequent test data. One way to overcome this overtraining problem is to stop the training early, which limits the magnitude of the weight vector below what it would be if the training were allowed to continue until a near-global training error minimum were found. The question then is when to stop the training. Here, the relationship between training data set size and the magnitude of the weight vector providing good generalization results is empirically established using cross-validation analysis on small subsets of the training data. These results are then used to estimate at what weight vector magnitude the training should be stopped when using the full data set. The general classification ability of an ANN trained in this manner is shown to increase the percentage of correctly classified test data points by an average of 1.5% over that of one trained using true cross-validational early stopping on a smaller data set. The technique of hyperspherical backpropagation, which entails training at a set weight vector magnitude, is also introduced and shown to be useful in lowering the validation error during training.

Table of Contents

	Page
List of Figures	v
Abstract	i
I. Introduction	1-1
1.1 Problem Statement	1-1
1.2 Scope	1-3
1.3 Contributions	1-4
1.4 Organization	1-5
II. Background	2-1
2.1 Pattern Classification	2-1
2.1.1 ANNs for Classification	2-1
2.2 Weight Initialization	2-6
2.3 Searching for the Minimum Error	2-7
2.3.1 Genetic Approaches	2-8
2.4 Generalization	2-13
2.4.1 Radial Complexity	2-15
2.4.2 Regularization	2-20
2.4.3 Bayesian Analysis for Classification	2-22
2.4.4 Cross-Validation	2-27
2.4.5 Magnitude of the Weight Vector	2-28
2.5 Summary	2-30
III. Achieving Good Generalization	3-1
3.1 Initial Radial Complexity	3-1
3.1.1 Weights Distributed Normally	3-2

	Page
3.1.2 Weights Distributed Uniformly	3-5
3.2 Consistent Behavior of Radial Complexity During Training . . .	3-10
3.2.1 Consistency of Training Behavior When Using EP Training	3-18
3.2.2 Consistency of Training Behavior When Training on TESSA Data Set	3-20
3.3 Cross-Validation Radial Complexity Estimation	3-21
3.3.1 Generalization Error	3-22
3.3.2 Early Stopping at the Estimated Radial Complexity . .	3-24
3.4 Training at a Fixed Radial Complexity	3-29
3.4.1 Genetic Approach	3-29
3.4.2 Hyperspherical Backpropagation Approach	3-30
3.5 Summary	3-35
IV. Conclusions and Recommendations	4-1
4.1 Conclusions	4-1
4.2 Recommendations for future Research	4-2
Appendix A. Weight Update Formula	A-1
A.1 Standard Batch Backpropagation	A-1
A.1.1 Second Layer Weight Update	A-1
A.1.2 First Layer Weight Update	A-2
A.2 Weight Updates with Regularization	A-3
Appendix B. Hyperspherical Coordinate Transformation	B-1
Appendix C. Derivation of Angle Updates for Hyperspherical Backpropagation	C-1
C.1 Second Layer Angle Update	C-1
C.2 First Layer Angle Update	C-3
Bibliography	BIB-1

List of Figures

Figure		Page
1.1.	Hand-written Characters	1-3
1.2.	Infrared Image	1-4
2.1.	Activation Function	2-2
2.2.	ANN Example	2-3
2.3.	Genetic Algorithm	2-10
2.4.	Sequential Squashing	2-12
2.5.	Sequential Squashing Demo	2-14
2.6.	Sigmoid Behavior	2-17
2.7.	Discriminant Boundaries	2-19
2.8.	Regularization Effect	2-21
2.9.	Bayesian Training	2-26
2.10.	Cross-Validation Example	2-28
3.1.	Decay of the variance of the radial complexity with increasing W . . .	3-10
3.2.	Growth of the expected value of the radial complexity squared and the square of the expected value of the radial complexity with increasing W	3-11
3.3.	Growth of the approximate expected value of the radial complexity and the average observed magnitude of the radial complexity with increasing W	3-12
3.4.	SSE and Standard Backprop	3-14
3.5.	SSE and Bayesian Backprop	3-15
3.6.	Softmax error and Standard Backprop	3-16
3.7.	Softmax error and Bayesian Backprop	3-17
3.8.	EP for ANN Training	3-18
3.9.	IR training	3-20
3.10.	Cross-Validation result for 5 training data points from each class. . . .	3-25

Figure		Page
3.11.	Cross-Validation result for 50 training data points from each class. . .	3-26
3.12.	Estimation of radial complexity	3-27
3.13.	Cross-Validation result for 100 training data points from each class. . .	3-27
3.14.	Overtraining on the TESSA data	3-28
3.15.	Using an EP to train the ANN at a specific radial complexity.	3-29
3.16.	Standard backpropagation versus hyperspherical backpropagation on OCR data	3-34

Radial Complexity Estimation for Improved Generalization in Artificial Neural Networks

I. Introduction

1.1 Problem Statement

Artificial Neural Networks (ANNs) are approximation methods of establishing a relationship between an input vector, \mathbf{x} , and an output vector, \mathbf{y} . The information about this relationship is stored in a set of scalars called weights. In a feed-forward ANN, this relationship is usually approximated by “training” the weights with a set of training vectors. The two most important aspects of training an ANN are the convergence speed and the ability to generalize well [69]. A significant amount of effort has gone into speeding up the training time of an ANN [15, 36, 37, 48, 49, 66, 71, 75, 78, 79]. Of the two, though, the generalization ability of the network will determine its applicability to a given problem after training [7, 59]; an ANN which does not generalize well will quickly start to “gather dust” since it does not perform consistently for new input data. This good generalization capability can be achieved a number of ways, including early stopping [7], pruning/growing of the hidden layer nodes [51], cross-validated early stopping [59], and regularization [8, 56] (including Bayesian methods [10, 41]). These methods attempt to limit the *effective complexity* of the network, the effective complexity being the ability of the network to capture the underlying structure of the training data. If the effective complexity is too low, the network cannot model the underlying structure of the training data well so the error will be consistently high on the training data and any test data. If the effective complexity is too high, the network begins to *memorize* the training data, including any noise in regression problems and outliers in classification problems. This yields a low error on the training data, but will tend to yield inconsistent error on subsequent test data. In neither case is good

generalization observed, since the network is either insufficiently complex (too general) or too complex to be general enough to give consistent, low-error classification results on future data.

The pursuit of good generalization should determine the network architecture and the magnitudes of the scalar weights. The complexity of an ANN necessary to achieve good generalization for a given problem is determined in part by the size of the training set [7,83]. This means that if we split our data into separate sets for training, cross-validation, and testing, we are limiting ourselves to an effective complexity that is smaller than one that would be allowed if we used all available data for training [59]. There are three factors which constrain the complexity of the discriminant boundaries when using an ANN as a pattern classifier: the number of sigmoid building blocks ($S1$), the magnitude of the weight vector (ρ), and the training data (amount available and intrinsic complexity of the distribution). These three factors work in conjunction with each other, so, for example, if there are N training examples, there will be some optimal $S1$ that will prevent overtraining regardless of the magnitude of the weight vector, ρ . This is known as structural stabilization. Or, for a given $S1$ and N , we can limit ρ such that overtraining will not occur. Ideally, though, we want to constrain the complexity using the training data to as great a degree as possible since this is the best information available about the distribution of the inputs which determines the classification boundaries in a Bayes optimal classifier [7,16]. The more training data we have, the closer we can build our discriminant functions to the Bayes optimal discriminant functions. The problem, then, is to use as much training data as possible to train the ANN so that we approximate the Bayes' optimal discriminant function as closely as possible. Unfortunately, when training set size is finite, the training data alone frequently will not provide adequate constraint of the complexity of the discriminant boundaries to prevent overtraining and the resulting network does not perform optimally when classifying new data. In this case, we then need to limit the number of hidden nodes, $S1$, or the magnitude of the weight vector, ρ . Bartlett has argued that limiting ρ is more important than limiting $S1$ since a larger number of sigmoid building blocks (quantified by the number of hidden nodes, $S1$) can provide a closer approximation to the Bayes optimal discriminant boundaries [4].

In the past, the ANN training method of early stopping based on the cross-validation error has proved somewhat successful, but has not taken advantage of the full training data set so as to best approximate the Bayes optimal discriminant boundaries [16]; while methods of ANN training that use the full data set have not necessarily provided good generalization capabilities upon completion of training [7].

1.2 Scope

This research is limited to feed-forward single-hidden-layer ANNs. Batch backpropagation is used since this method is theoretically guaranteed to converge to a solution [7,59]. The data sets used include a set of hand-written numerical characters from 0 – 9 (OCR data set), an example of which is shown in Figure 1.1, as well as a set of infrared image data (TESSA data set), an example of which is shown in Figure 1.2. These two data sets are

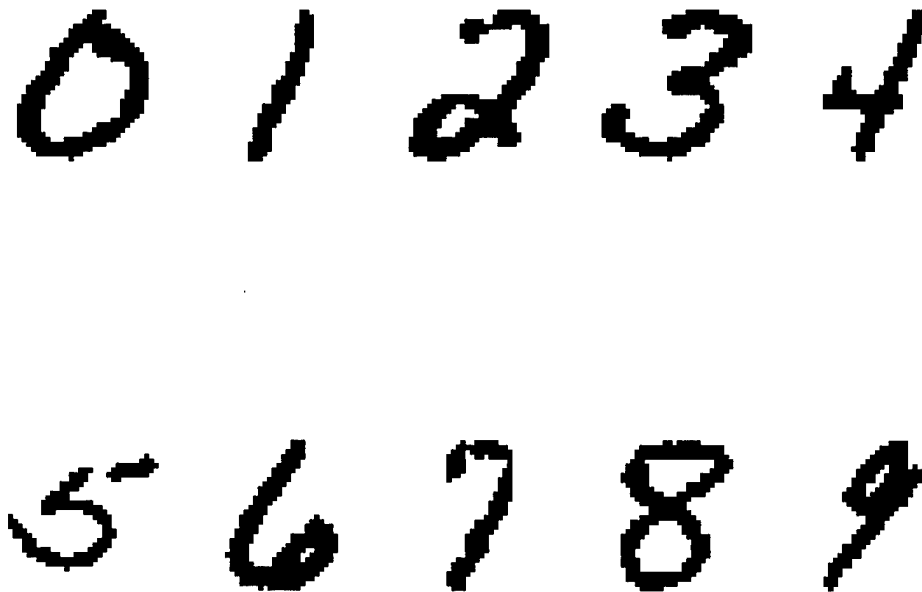


Figure 1.1 Example of hand-written characters, or the OCR data set.

representative of the types of data ANNs are used to analyze in the real world, with the TESSA data set being particularly difficult to classify.

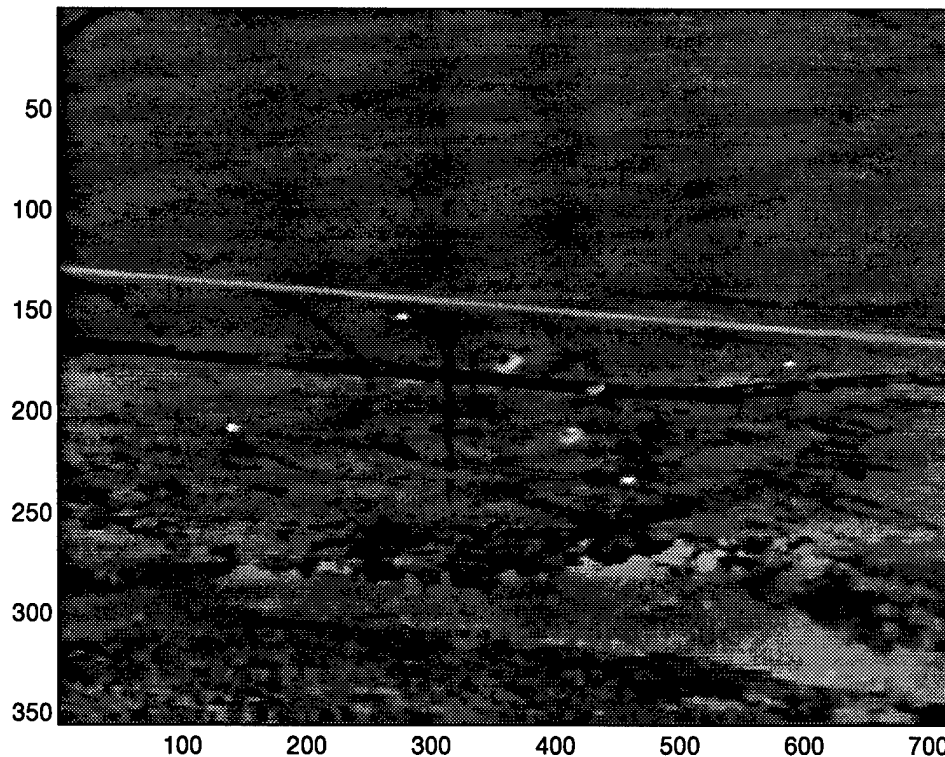


Figure 1.2 Example of an infrared image from the TESSA data set.

1.3 Contributions

Several contributions are presented to the field of Artificial Neural Networks. First, the expected initial “radial complexity,” defined as the magnitude of the weight vector, is derived for the case of the individual weights being initialized by drawing random variables from uniform and normal distributions. Work in the past has concentrated primarily on initializing the weights so as to decrease training time, while the consideration here is for improving the generalization ability. Second, the radial complexity is shown experimentally to behave consistently during training from run to run. This behavior justifies the cross-validational early stopping procedures used here and in previous work. Third, the procedure of “radial complexity estimation” which allows the weights to be trained based on the magnitude of the weight vector is developed. Using this estimated radial complexity is shown to lead to improvements in classification ability on data sets which are prone to overtraining. Finally, the method of “hyperspherical backpropagation,” is developed and shown to lead to lower error on the validation set during training.

1.4 Organization

Chapter 1, Introduction, explains the problem to be solved, defines and limits the scope of the research, and presents the contributions to the field. Chapter 2, Background, reviews the current literature on methods of achieving good generalization in multi-layer feed-forward ANNs, including discussions on the effect that the radial complexity has on the decision boundaries. Chapter 3, Achieving Good Generalization, derives the expected value of the radial complexity during weight initialization and demonstrates the feasibility of using pseudo-cross-validated early stopping based on an estimated radial complexity to achieve good generalization in a multi-layer feed-forward ANN using data sets which exemplify real-world classification problems. Also, the subject of hyperspherical backpropagation is developed and shown to improve the validation error during training. Finally, Chapter 4, Conclusions and Recommendations, summarizes where this research puts the field and where further research is indicated. The following chapter describes previous research in the area of using ANNs for pattern recognition.

II. Background

Man is constantly trying to teach machines to ease his workload. Some types of pattern recognition are considered quite overwhelming or tedious for a human; analyzing large numbers of mammograms for possible cancer [86], determining the identity of a suspect based on comparing fingerprints with those on file in a huge database [25], or recognizing a particular phoneme in a set of speech signals [80]. In this chapter, the performance of ANNs as pattern recognizers is discussed, as are the steps necessary to assure that trained ANNs perform well when making decisions after training. The ideal pattern classifier is the Bayes optimal classifier since it provides the minimum probability of misclassification.

2.1 Pattern Classification

The best error rate one can hope to consistently achieve in any classification problem is the Bayes error rate. This is the error achieved when using a Bayes optimal classifier, which uses the distribution of the inputs to make classification decisions [7, 16] and minimizes the probability of misclassification by using the posterior probability

$$P(C_k|\mathbf{x}) = \frac{p(\mathbf{x}|C_k)P(C_k)}{p(\mathbf{x})}$$

to make classification decisions. Unfortunately, the true statistical properties of the input data are seldom known, so various methods are employed to mimic the Bayes optimal classifier. After training an ANN as a pattern classifier, the best generalization results are attained if it forms Bayes optimal decision boundaries.

2.1.1 ANNs for Classification. Historically, maximum likelihood estimation (MLE) techniques, such as backpropagation of error, have been very popular for training neural networks. Several sources [7, 59, 60] give excellent treatments on these methods. Ruck demonstrated that when training an ANN as a pattern classifier using Sum-square error (SSE), upon completion of training it provides a good approximation to a Bayes optimal classifier [61]. SSE is the most widely used criterion for evaluating the error of a network,

but other error functions can be used (i.e. Minkowski error), and, in fact, for ANNs used for classification, the softmax error is more appropriate [7] for approximating the posterior probability of an input belonging to a specific class. The outputs of the network can be interpreted as probabilities of class membership if we structure our network using logistic sigmoid activation functions (see Figure 2.1) for the hidden layer and softmax activation functions (a generalization of the logistic sigmoid activation function) for the output layer [7, 9, 59]. Using the softmax function allows us to interpret the outputs of the network as

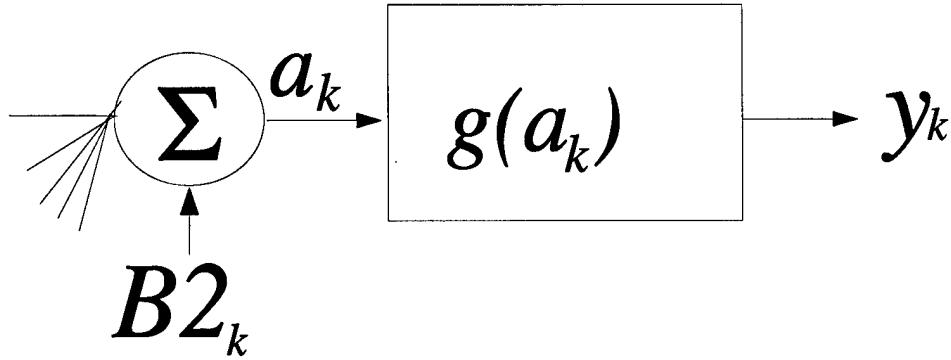


Figure 2.1 Demonstration of how the activation function, $g(a_k)$, is related to the output of a given node.

probabilities of class membership by forcing the values at the output layer of the network to lie in the range (0,1) and sum to 1. The softmax function is defined as

$$y_k = g(a_k) = \frac{\exp(a_k)}{\sum_{k'} \exp(a_{k'})}. \quad (2.1)$$

The summation over k' is over all the outputs and acts as the normalization factor.

Consider a classification problem on the well-known IRIS data set. This data contains three classes of flowers and each data vector consists of four features. The database has 150 input feature vectors. An example ANN architecture used to classify this data is shown in Figure 2.2. This ANN has two layers and five hidden nodes. The weights feeding into the hidden layer are denoted as $W1_{ji}$, the biases feeding into the hidden layer are denoted as $B1_j$, the weights feeding into the output layer are denoted as $W2_{kj}$, and the biases feeding

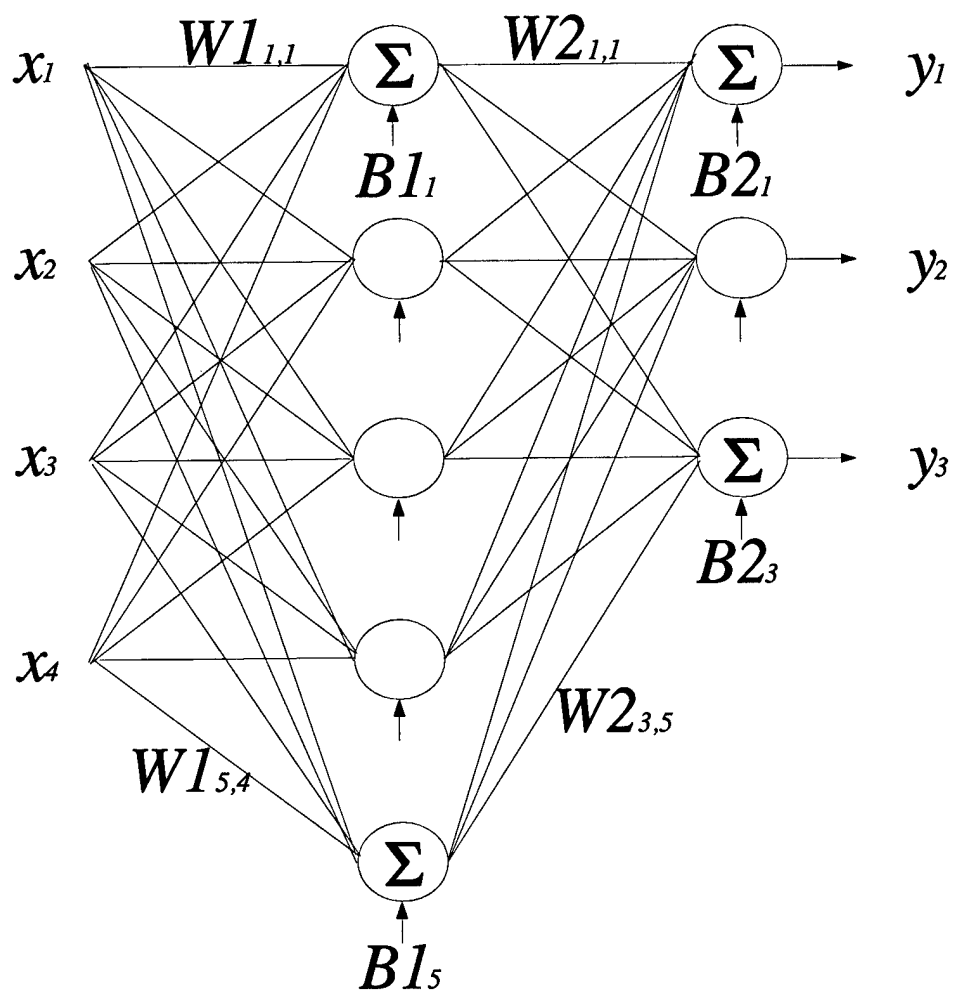


Figure 2.2 Artificial Neural Network used in our example.

into the output layer are denoted as $B2_k$. The form of the weight vectors is,

$$\begin{aligned}
 W1 &= \begin{bmatrix} W1_{1,1} & W1_{1,2} & \cdots & W1_{1,R} \\ W1_{2,1} & W1_{2,2} & & W1_{2,R} \\ \vdots & & \ddots & \\ W1_{S1,1} & W1_{S1,2} & & W1_{S1,R} \end{bmatrix}, \\
 B1 &= \begin{bmatrix} B1_1 \\ B1_2 \\ \vdots \\ B1_{S1} \end{bmatrix}, \\
 W2 &= \begin{bmatrix} W2_{1,1} & W2_{1,2} & \cdots & W2_{1,S1} \\ W2_{2,1} & W2_{2,2} & & W2_{2,S1} \\ \vdots & & \ddots & \\ W2_{K,1} & W2_{K,2} & & W2_{K,S1} \end{bmatrix}, \\
 B2 &= \begin{bmatrix} B2_1 \\ B2_2 \\ \vdots \\ B2_K \end{bmatrix}.
 \end{aligned}$$

With four input features, five hidden nodes, and three classes, we have $4 \times 5 + 5 \times 3 = 35$ weights as well as $5 + 3 = 8$ biases for a weight space of dimension 43. The final weight vector, after training, would ideally give an output for a set of training data that had low error between the target vectors and the output vectors, while also yielding low error on the test data. Here, the training data set is denoted by

$$D = \{(\mathbf{x}^{(1)}, \mathbf{t}^{(1)}), \dots, (\mathbf{x}^{(N)}, \mathbf{t}^{(N)})\}, \quad (2.2)$$

with $\mathbf{x}^{(n)}$ being the n^{th} training vector and $\mathbf{t}^{(n)}$ the n^{th} target vector that represents the class membership of the n^{th} training vector. A typical normalized feature vector is

$$\mathbf{x}^{(1)} = \begin{bmatrix} -0.8977 \\ +1.0286 \\ -1.3368 \\ -1.3086 \end{bmatrix}.$$

This feature vector belongs to class 1, so

$$\mathbf{t}^{(1)} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}.$$

This network has three outputs: output *one* is the probability that the input, $\mathbf{x}^{(n)}$, is a member of class one, C_1 , given the training data, D , and outputs *two* and *three* are the probabilities of belonging to class two and three, respectively, such that

$$\begin{aligned} y_1^{(n)} &= P(\mathbf{x}^{(n)} \in C_1 | D), \\ y_2^{(n)} &= P(\mathbf{x}^{(n)} \in C_2 | D), \\ y_3^{(n)} &= P(\mathbf{x}^{(n)} \in C_3 | D). \end{aligned}$$

When using the softmax function at the outputs, the error function used for classification takes the form

$$E_D = - \sum_{n=1}^N \sum_{k=1}^K t_k^{(n)} \ln(y_k^{(n)}) \quad (2.3)$$

which is based on Bishop's cross-entropy for multiple, mutually exclusive classes [7]. $t_k^{(n)}$ is the target value at output k for input vector n , while $y_k^{(n)}$ is the actual value at output k for input vector n , and K denotes the number of disjoint classes, (C_1, C_2, \dots, C_K) . For completeness, we derive the weight update procedure for this error function in Appendix A.

Upon first initializing the weights, the output, $\mathbf{y}^{(1)}$, for input vector $\mathbf{x}^{(1)}$ shown above is

$$\mathbf{y}^{(1)} = \begin{bmatrix} 0.3443 \\ 0.3178 \\ 0.3379 \end{bmatrix}.$$

This equal probability state makes sense since our weights are still in a random state and no training has happened yet. After just five training epochs (weight updates) through the training data set, D , the output vector for our class one training input vector is

$$\mathbf{y}^{(1)} = \begin{bmatrix} 0.9634 \\ 0.0000 \\ 0.0366 \end{bmatrix},$$

which is much closer to the desired output of

$$\mathbf{t}^{(1)} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}.$$

We interpret this output as the probability of the input vector belonging to class 1 is approximately 0.96, or

$$P(\mathbf{x}^{(n)} \in C_1 | D) \approx .96.$$

The training algorithm pushes the weight vector into the direction necessary to lower the error on the training set, but what then should be the starting point of this weight vector?

2.2 Weight Initialization

The importance of the initial weight vector is an often overlooked part of the training process [29]. In the past, for standard backpropagation, the approach to choosing a starting weight set has been to choose initial weights from a uniform distribution between plus and

minus α , usually with $\alpha = .5$ [29]. Kolen argues that the magnitude of the weight vector at initialization plays a key role in the convergence speed of the backpropagation algorithm [29]. Other methods of weight initialization have also focused primarily on speeding up the weight training process [48, 64], although Denoeux uses prototypes for weight initialization and considers the repercussions that this initialization can have on generalization [13]. Bayesian backpropagation relies on a “prior” probability distribution of the weights which is usually chosen to be a normal distribution with a parameter, α , governing the variance of that distribution [7, 10, 41, 59]. α is chosen based on our prior belief on how closely we think each weight is to zero. Although the point of regularization is to create a better generalized ANN, there has not previously been a direct relationship established between α and the ability of the network to generalize well.

After initialization, the weights are usually trained using backpropagation of error, but does the resultant set of weights provide the lowest possible error on the training set?

2.3 Searching for the Minimum Error

With MLE techniques, such as back-propagation of error, the error is computed as a function of the weights and some gradient descent technique is used to find a local minimum. Though this takes account of only one of many possible minima in weight space, the results are frequently satisfactory enough to justify our limited search. If the network does not converge to an adequate solution, the standard procedure is to restart the algorithm with a new random set of weights [12] to find a more suitable solution.

Fogel [18] points out that one popularly accepted disadvantage of many MLE techniques (such as gradient descent along the error surface) is the propensity for the weight vector to become trapped in a local error minimum that is unsuitable. Although these methods converge to a solution quickly, once perceived to be trapped in an unacceptable local minimum the algorithm is usually reinitialized with a random weight vector and the training restarted [12]. Much effort has gone into finding the global error minimum when training the weights [34, 82], but Lawrence has recently cast aspersions upon this procedure [33]. He points out that the minimum error found by gradient descent methods when

training an ANN is often significantly worse than the global minimum error. This makes sense when considering generalization ability of an ANN; while low error is obviously a desirable characteristic, the weight vector yielding the lowest error obtainable on the training set will almost never be the weight vector yielding the lowest error on the test set. This subject will be further expanded in section 2.4. Other methods exist for training ANNs, one such being based on evolution in living organisms.

2.3.1 Genetic Approaches. Another method of determining the weights in an ANN is by letting the weights evolve over time in such a way as to mimic the evolution of an organism. These genetic approaches have become more popular for searching out local error minimums in ANNs [27, 34, 47, 53, 67, 69, 81, 84].

2.3.1.1 Genetic Algorithms. Genetic Algorithms (GAs) have been used to determine weights in neural networks with varied success [28, 30, 65]. GAs are loosely based on models of genetic change, or evolution, in populations of individual organisms [22]. Each organism (weight vector) is defined as a chromosome, which in turn is made up of some pre-determined number of genes (bits representing weights). These genes are often treated as binary values, so a typical chromosome would be represented by a string of genes, or vector, such as $(100110000111010...11001)^T$. The fitness of each of these organisms can be measured, and possible goals include invoking an evolutionary process to either improve the overall fitness of the population or to obtain a highly fit single member. This idea of fitness governs the extent to which an individual organism can influence future generations, and genetic operators have been developed to propagate this influence. Crossover and mutation are the operators most often used, where crossover is the swapping between two chromosomes of some subset of their genes, and mutation is the bit-switch of randomly selected genes in a chromosome. Crossover allows organisms to evolve much more rapidly than they would if each offspring simply contained a copy of the parent chromosome, occasionally modified by a mutation, and corresponds to a large step size in our weight space. Mutation, on the other hand, offers the opportunity for new genetic material to be introduced into the population, producing a more robust search of the entire solution space, and mutation corresponds to a

small step size in our search over weight space. A typical GA search is described below and illustrated in Figure 2.3.

2.3.1.2 Typical GA Search.

1. Randomly generate an initial population of chromosomes.
2. Test the fitness of each chromosome and save the single chromosome which is most fit in "most-fit" queue.
3. Generate a new population from the old population using fitness of members of old population and a "roulette wheel" random sorting with greater fitness increasing the probability of being picked.
4. Perform crossover and mutation over entire new population.

Each chromosome has a random chance for crossover. When tagged, a chromosome will interchange some subset of its genes with the same subset of genes in another tagged chromosome, for example

$$\left\{ \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \right\} \Rightarrow \left\{ \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \right\}. \quad (2.4)$$

Each gene has a random chance for mutation. When tagged, a bit switch occurs (i.e., $1 \rightarrow 0$ and $0 \rightarrow 1$).

5. Test the fitness of each chromosome in the new population and save the single fittest chromosome if it is more fit than the chromosome currently in the most-fit queue.
6. The new population takes the place of old population and loops back to step 3.
7. Continue until some termination criteria is met (high fitness, number of generations, etc.).
8. Chromosome in most-fit queue is solution.

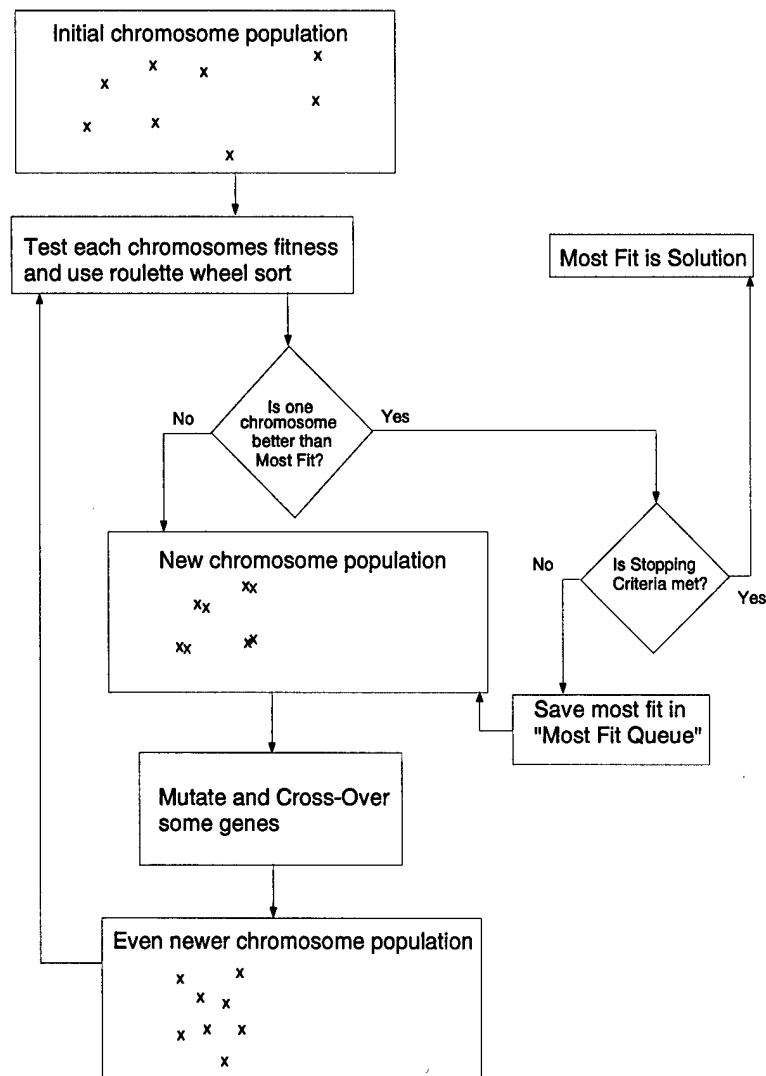


Figure 2.3 Typical Genetic Algorithm Search [23].

Korning points out that training weights in an ANN requires very large chromosomes to ensure sufficient dynamic range for the magnitudes of the weights [30]. Angeline [3]

gives good reasons to believe that GAs are less than optimal for training neural networks, especially when compared with the more general Evolution Program.

2.3.1.3 Evolution Programs. Evolution Programs (EPs) are a generalization of Genetic Algorithms. While Genetic Algorithms are generally considered to be limited to binary representations of the data, Michalewicz points out that EPs use whatever form is most useful (usually a form closely related to the actual data) to solve a given problem [43]. A typical chromosome may then be simply a vector of real numbers. With EPs, we are not limited to or bound by the crossover and mutation operators typically used in GAs. If mutation and crossover are used, and if real numbers replace binary, possible solutions pointed out by Michalewicz are:

1. For crossover, simply swap some subset of genes (now real numbers instead of bits) as in GA crossover.
2. For mutation, replace the tagged gene with a new real number generated by some probability distribution.

EPs have been used to find the weights in ANNs [3,57,68]. Fogel argues that mutation is the dominant operator in Evolutionary Programming [18], and Angeline points out that crossover may be particularly inappropriate when training weights for a neural network [3]. Porto [57] uses only a mutation operator and a fitness function based on the sum-squared error. He perturbs chromosomes with a normal random variable whose variance is equal to that error, at which point new and old chromosomes compete to find a new population that has an overall lower error. EPs and GAs, though, converge to single solutions; when more than one solution is desired, we need an algorithm that can find multiple solutions.

2.3.1.4 Sequential Niche Techniques. Evolution Programs are fundamentally tools to solve maximization problems, finding a local maximum of a function using appropriate genetic operators. Beasley [5] developed a method that allows conventional EP techniques to be used to find an arbitrary number of local maxima of a function with several local maxima. This method is outlined in Figure 2.4 and summarized as follows [5]:

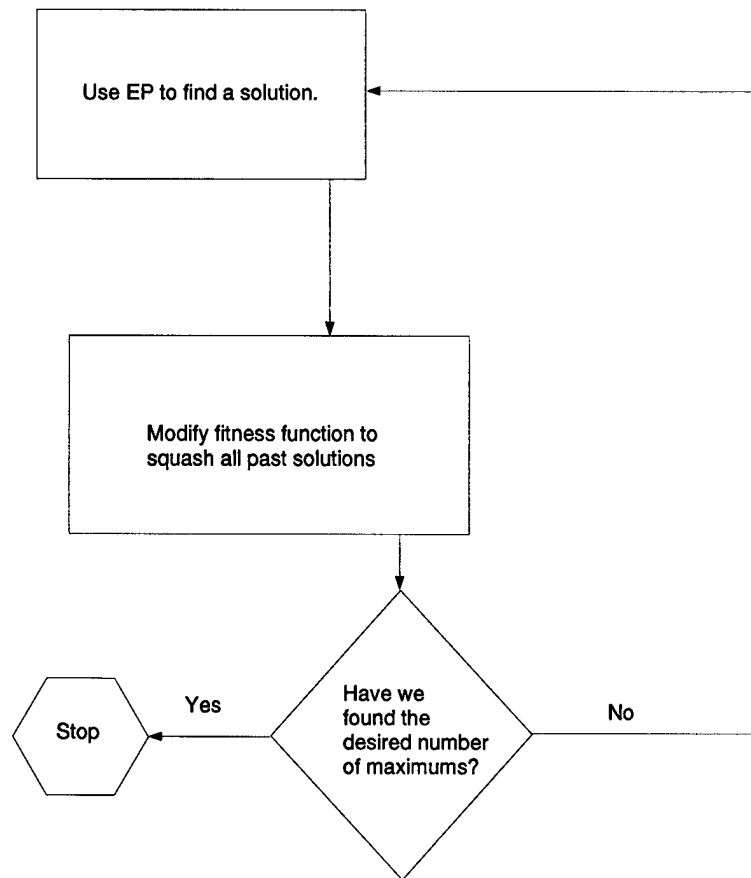


Figure 2.4 Illustration of Sequential Squashing of the Fitness Function [5].

Sequential Niche Algorithm.

1. Find an initial local maximum using any standard EP technique.
2. Modify the fitness function in the region surrounding the initial local maximum using a “squashing” function to eliminate it from subsequent searches.
3. Search for as many additional local maxima as desired, re-modifying the fitness function after each solution is obtained to eliminate that solution from the search. The locations of all local maxima are stored in memory and the fitness function is modified to squash all local maxima found in previous searches.

Beasley discusses a number of squashing functions, one of which is

$$G(r) = \begin{cases} m^{1-d_{xs}/r} & \text{if } d_{xs} < r, \\ 1 & \text{otherwise} \end{cases} . \quad (2.5)$$

where $m > 0$ is the desired multiplicative factor at the center of the solution on subsequent fitness tests, r is the user-defined radius about the solution that will be affected by G , and d_{xs} is the actual distance of the chromosome from the solution mode.

Figure 2.5 demonstrates the application of Equation (2.5). Here, we have a four maxima fitness function in two-dimensional space. As the search locates each maximum, the fitness function is modified to eliminate that maximum from the search until all four maxima of the function are found.

Having discussed some of the most widely accepted methods for training ANNs, we need to recall that the primary concern of any ANN design should be that the network generalizes well when tested on previously unseen data.

2.4 Generalization

An ANN's usefulness after training is completely determined by its ability to correctly analyze future data generated by the same process that generated the training data [1,

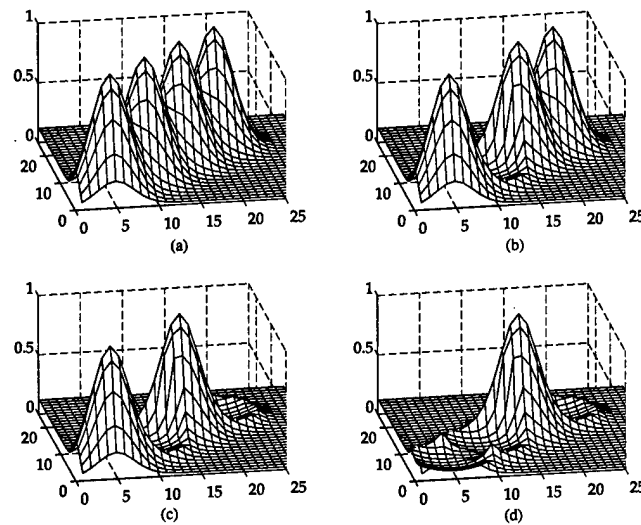


Figure 2.5 Plot (a) shows the unaltered, initial fitness function. Plot (b) shows the altered fitness function after one of the maxima has been found by the EP and subsequently squashed. Plots (c) and (d) show finding and squashing of the remaining maxima and how this affects the fitness function each time.

11, 31, 33, 35, 44, 50, 54, 58, 62, 76, 85]. A number of researchers have bounded the expected generalization error [21, 42], and some attempt to estimate the generalization ability of an ANN after training [52, 63, 73, 74]. It is well known that reducing the complexity of the network leads to better generalization [14]. Reducing the complexity entails limiting the architecture or limiting the value of the individual weights [19, 24]. Efforts to achieve good generalization by limiting the number of free parameters, or structural stabilization, have been carried out using growing and/or pruning of weights and hidden nodes [14, 51].

There are a number of ways to limit the value of the individual weights so as to improve generalization. Regularization appends a function of the weights to the error function so as to penalize large weight vector magnitudes during training, while cross-validated early stopping bases the stopping criteria on the validation set classification error rather than on the training set classification error, thereby stopping training while the weight vector's magnitude is smaller than it would be if only the training set error was taken into consideration [7, 59]. Some research investigates adding noise to the training data so as to "smear out" the data and make overtraining less likely [26, 72], but Bishop indicates that

this method is closely related to limiting the magnitude of the weights in much the same way as regularization [7]. Another form of limiting the value of the weights is soft weight sharing. Once again, though, Bishop points out that this is another form of regularization.

The magnitude of the weight vector is central to the issue of good generalization and is here referred to as the “radial complexity.” The next section gives an example showing how changing the magnitude of the weight vector affects the decision boundaries for classification.

2.4.1 Radial Complexity. The fact that the magnitude of the weight vector is important for good generalization has been used in regularization by many researchers [7, 59], and Bartlett has pointed out that the size of the individual weights (which is directly correlated with the magnitude) is more important than the number of weights in problems where ANNs are useful [4]. Dunne analyze the evolution of the weights in a simple, two feature, two weight, problem as the polar coordinate characteristics of the weights were varied [17]. To understand how the magnitude of the weight vector affects the classification ability of the ANN trained as a pattern classifier, we need to understand how an ANN constructs discriminant boundaries from summations of weighted sigmoids. There are many factors which affect how the decision boundary is constructed. First, let us look at the equation for the output of a hidden node,

$$z_j = g(w_{j1}x_1 + w_{j2}x_2 + \dots + w_{jR}x_R + b_j), \quad (2.6)$$

where w_{ji} is the weight from input x_i to hidden node j , b_j is the bias feeding into node j , and $g(\cdot)$ is the logistic sigmoid output activation function

$$g(a) = \frac{1}{1 + e^{-a}}. \quad (2.7)$$

Another sigmoid activation function often used is the tanh activation function,

$$g(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}, \quad (2.8)$$

but is related to the logistic sigmoid through a linear transformation (which can be done by the weights and biases), so the choice of which to use is irrelevant at the hidden layer [7]. Each hidden node contributes a sigmoid which is the basic building block of the discriminant boundaries.

The logistic sigmoid is a non-decreasing function and is approximately linear in the region around $a = 0$. If we consider this region then we see that the output of the hidden node (as a function of the input) lies on a hyperplane whose slope is determined by the weights [17] and whose position is determined by the bias term. Where this hyperplane crosses through $g(0)$ forms a preliminary decision boundary. Therefore, we can express this individual decision boundary as

$$w_{j1}x_1 + w_{j2}x_2 + \dots + w_{jR}x_R + b_j = 0. \quad (2.9)$$

For simplicity, let us consider the two-feature case. The extension into multiple dimensions is straightforward. Now

$$w_{j1}x_1 + w_{j2}x_2 + b_j = 0. \quad (2.10)$$

So, if $w_{j2} \neq 0$, then

$$\begin{aligned} x_2 &= -\frac{w_{j1}x_1 + b_j}{w_{j2}} \\ &= -\frac{w_{j1}}{w_{j2}}x_1 - \frac{b_j}{w_{j2}}. \end{aligned} \quad (2.11)$$

From Equation (2.11), we can see that the slope and intercept of the decision boundary are functions of the *ratio* of weights. Changing the magnitude of a vector containing these three weights has no effect on the location of this individual sigmoid in input space, but does change the slope of the hyperplane approximation in the linear region of the logistic sigmoid. Figure 2.6 demonstrates how changing a weight's magnitude changes the slope of the sigmoid in the linear region. Keep in mind, though, that a given output, y_k , is determined by a function that is the shifted weighted sum of all the sigmoids from all the hidden nodes

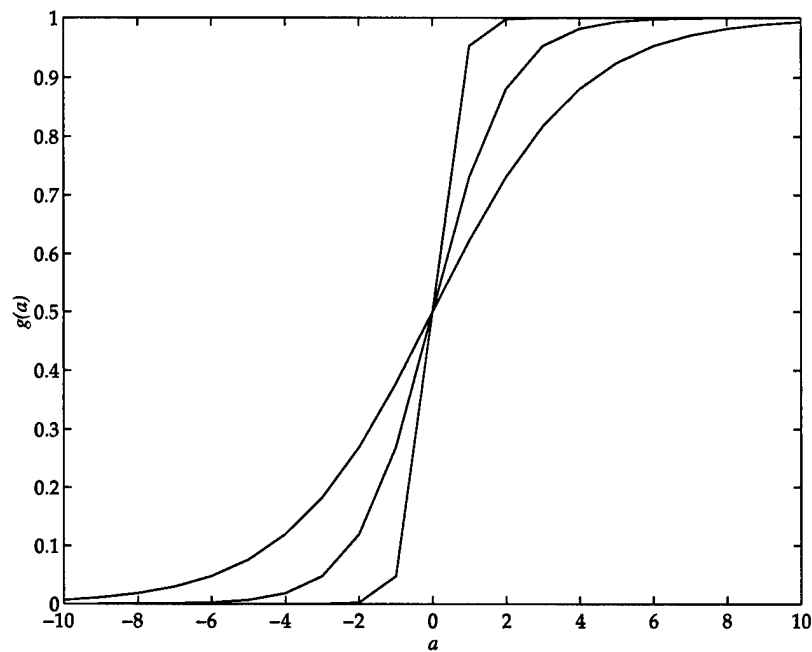


Figure 2.6 Demonstration of how the slope of the linear region of the sigmoid function (the region around $g(0)$) changes with changing weight magnitude. The steeper slope corresponds to a larger weight magnitude while a shallower slope corresponds to a lower weight magnitude.

so that

$$y_k = g(w_{k,1}z_1 + w_{k,2}z_2 + \dots + w_{k,S1}z_{S1} + b_k), \quad (2.12)$$

where $w_{k,j}$ is the weight going from hidden node j to output node k , and b_k is the bias feeding into node k . The form of the activation function at each output node is important in interpreting each output's meaning for training, but is not the primary factor affecting the decision boundaries in the input space. Each weight going into the output layer again alters the slope of the approximation hyperplane used to build each individual decision boundary. The output bias translates all the decision boundaries, but affects all individual sigmoids used to create the decision boundaries equally. The primary factor, then, in the determination of the decision regions is the *ratio* of weights and biases, not the actual values. The importance in the actual values of the weights is seen when looking at the summation of the sigmoids. Where the sigmoids intersect forms a "corner," and the "sharpness" of this corner depends on the steepness of the two sigmoids; two steep sigmoids intersecting will form a sharp corner, while two shallow sigmoids meeting will have a more rounded corner. Remember, the position of the sigmoid decision boundary is a function of the ratio of weights, while the steepness of the sigmoid is a function of the magnitude of weights. In order for a network to overtrain, it needs to "reach out and grab" outlying data points that lie within what should statistically be another class's decision space. To construct a decision region such as this, narrow and long, would require sharp corners. By limiting the magnitude of the weight vector, we have limited the ability of the ANN to construct sharp corners and therefore limited its ability to overtrain. Figure 2.7 shows the effect that changing the magnitude of the weight vector has on the decision boundaries. This figure shows contour plots of a triangular decision region formed from three sigmoids in two-dimensional space. Notice that decreasing the radial complexity has a "low-pass" filtering effect on the high frequency corners.

ANN training that concentrates on structural stabilization (limiting the number of weights) attempts to limit complexity by limiting the number of sigmoids used to build the discriminant functions, which we see now is the functional equivalent of limiting the magnitude of the weight vector since a smaller magnitude weight vector will require more

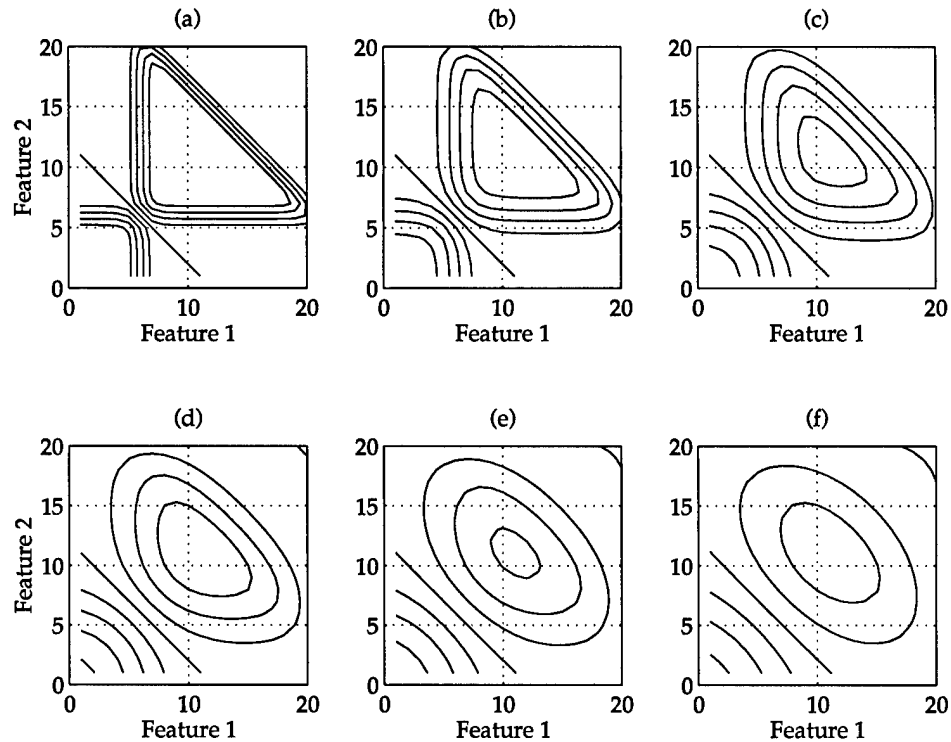


Figure 2.7 Example showing how changing the magnitude of the weight vector changes the ability of the ANN to make decisions based on discriminant boundaries. The magnitude of the weight vector decreases as we move from top left, plot (a), to bottom right, plot (f). Notice that the position of the discriminant boundaries remains unchanged, but the sharp corners are rounded off with a decrease in magnitude.

sigmoids to accomplish the same type of discrimination accomplished by fewer weights with unlimited magnitude. One way to keep the magnitude of the weight vector small is to use regularization.

2.4.2 Regularization. Regularization is a way to keep the magnitude of the weight vector relatively small so as to minimize over-training in a network and achieve good generalization [2, 7, 55, 56, 59]. The error function using regularization is the sum of the log likelihood error, E_D , and the regularization error, E_W ,

$$S(\mathbf{w}) = E_D + E_W \quad (2.13)$$

$$= - \sum_{n=1}^N \sum_{k=1}^K t_k^{(n)} \ln(y_k^{(n)}) + \frac{\alpha}{2} \|\mathbf{w}\|^2, \quad (2.14)$$

where

$$E_W = \frac{\alpha}{2} \|\mathbf{w}\|^2 = \frac{\alpha}{2} \sum_{i=1}^W w_i^2, \quad (2.15)$$

and $\alpha > 0$ is the *regularization coefficient*. Regularization effectively warps the error surface by adding it to a hyper-parabola centered at the origin, thus favoring weight vectors closer to zero. Figure 2.8 demonstrates the effect of regularization graphically. The weight update repercussions due to regularization are discussed in Appendix A.2. Ripley [59] feels that regularization of some sort should always be used when training an ANN. Notice that the effect of regularization is to decrease the effective complexity by decreasing the magnitude of the weight vector as the network is being trained.

The regularization coefficient used in regularization determines the amount that the weight vector's magnitude is penalized, and there is a direct relationship between this coefficient and the ability of the final network design to generalize well [7]. Larsen attempted to find the regularization coefficient that yielded the optimal generalization performance [32], while Bayesian backpropagation, discussed in the next section, allows this parameter to be changed while the network is being trained [7, 10, 41].

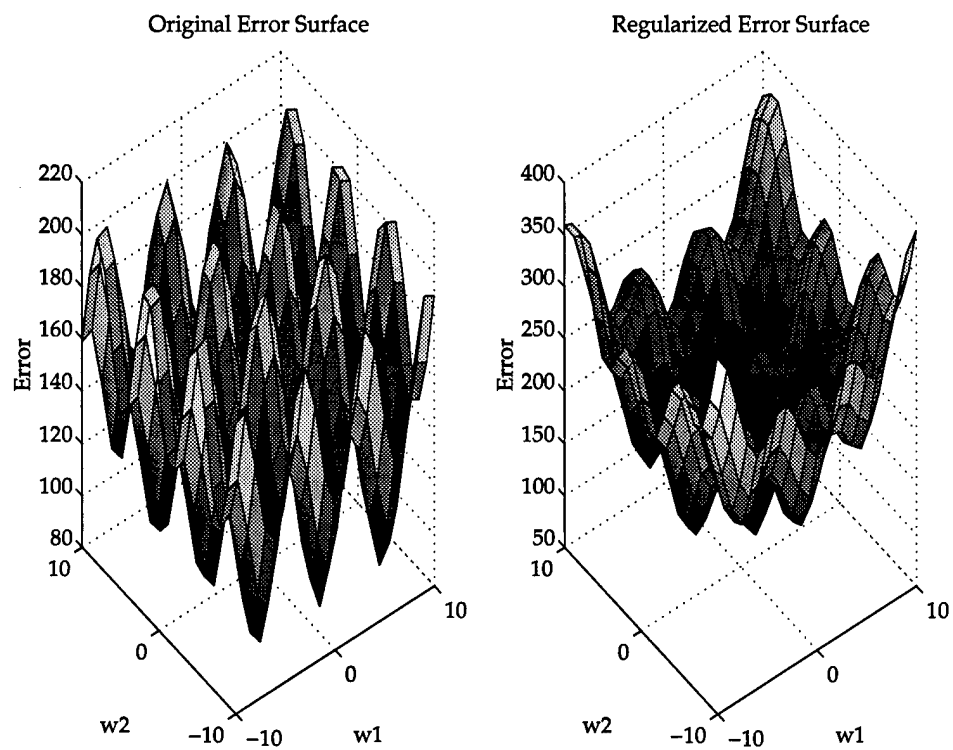


Figure 2.8 Effect of using the magnitude squared of the weight vector for regularization. Notice that the new error function is just the old error function warped to lie on the surface of a quadratic bowl.

Some forms of regularization attempt the use of the full data set as the training set, but standard regularization relies on the regularization coefficient, α , to determine the penalty placed on the magnitude of the weight vector. α can be chosen by using cross-validation to determine at what α overtraining occurs. This again, though, limits the size of the training data set used to determine α , thereby imposing too stringent a penalty on ρ and overconstraining the complexity of the discriminant boundaries that would be allowed if α could be determined using all the data.

Bayesian backpropagation is a form of regularization that attempts to overcome this overconstraining by using all available data when updating the regularization coefficient.

2.4.3 Bayesian Analysis for Classification. Recently, Bayesian techniques have been shown to be useful in terms of analyzing different aspects of neural network architecture [7, 10, 38–41, 46]. With Bayesian backpropagation, the regularization parameter α is updated during the training process. The limitations of this technique lie in the approximation of the error surface as a Gaussian function in the area local to the most probable weight vector, \mathbf{w}_{MP} , which yields lowest error on the training set. Here, though, there is no guarantee that the resultant weight vector provides the lowest generalization error. With this Gaussian approximation, the Hessian needs to be computed and its eigenvalues found in order to update α during training. A further approximation that avoids the evaluation of the Hessian actually uses the current magnitude of the weight vector to update α , thereby simply loosening the restrictions on the current weight vector's magnitude and carrying the training further from the search for an optimal generalization ability. These techniques have been used for (among other things) training the weights and choosing one network model over another. Rather than finding a local acceptable weight vector which minimizes regression or classification error, the Bayesian method (in its purest form) integrates over all weight space when calculating the output of the ANN. When discussing Bayesian techniques, *marginalization* becomes a topic of prime interest, since we need to integrate out the dependence of our answer on the weights. For example, the output of an ANN, in the strictest Bayesian sense, would be

$$\hat{y}^{(N+1)} = \int_{R^N} f(\mathbf{x}^{(N+1)}, \mathbf{w}) p(\mathbf{w} | D) d\mathbf{w}, \quad (2.16)$$

where $f(\cdot)$ is the function represented by the network, D is the training data set $D = \{(\mathbf{x}^{(1)}, \mathbf{t}^{(1)}), \dots, (\mathbf{x}^{(N)}, \mathbf{t}^{(N)})\}$, and $p(\mathbf{w}|D)$ is the probability density function. This integration considers the outputs resulting from all possible solutions in weight space weighted by the posterior distribution of the weights at those points. Therefore, final layer outputs resulting from weights that lie in an area of high posterior distribution will contribute more to the integration solution than outputs resulting from weights that lie in an area of low posterior distribution.

According to Bayes' Theorem [7], if $p(\cdot)$ is the pdf and $P(\cdot)$ is the cdf on \mathbf{R}^N , then

$$p(\mathbf{w} | D) = P(D | \mathbf{w}) \frac{p(\mathbf{w})}{P(D)}, \text{ for all } \mathbf{w} \in \mathbf{R}^N. \quad (2.17)$$

Notice that Bayes' Theorem can be interpreted as saying that the posterior distribution of the weights is equal to the probability of a data set being correctly classified given that set of weights (the likelihood), weighted by the ratio of the value of the weight prior density function at that point in weight space to the Probability of the data set.

When using the softmax function so that the outputs approximate the posterior probability of belonging to the correct class, the likelihood function is

$$P(D | \mathbf{w}) \simeq \prod_{n=1}^N \prod_{k=1}^C (y_k^{(n)})^{t_k^{(n)}}. \quad (2.18)$$

The likelihood function is the probability that all outputs are from the correct class (since $y_k^{(n)}$ is the actual output at node k and $t_k^{(n)}$ is the desired output at node k) and is a multiplication over all outputs and all data vectors from all the different classes.

The prior probability density function (pdf), $p(\mathbf{w})$, is based on our belief in the form the final weight vector should take. Usually, the bias/variance argument which affects the generalization ability of the network suggests that the solution weight vector will be relatively small [7], so $p(\mathbf{w})$ is usually chosen to be a Gaussian such that

$$p(\mathbf{w}) = \left(\frac{\alpha}{2\pi}\right)^{\frac{W}{2}} \exp\left(-\frac{\alpha}{2}\|\mathbf{w}\|^2\right). \quad (2.19)$$

This prior pdf imposes the same type of restrictions on the magnitude of the weight vector as does the regularization procedure discussed in Section 2.4.2. The regularization factor, α , is commonly referred to as a *hyperparameter* and is discussed in Section 2.4.3.1. Integrating over all weight space can prove somewhat impractical, so approximations are unavoidable.

2.4.3.1 Gaussian Approximation for Bayesian Training. MacKay [41], as well as Buntine and Weigend [10], applied the Bayesian approach to ANN training for practical applications. They make the assumption that the error surface in the vicinity of the “most probable” weight vector (one with lowest error over the training set), \mathbf{w}_{MP} , is locally a Gaussian function. This approximation allows the area in the vicinity of \mathbf{w}_{MP} to be evaluated as a quadratic error function, the analysis of which is straightforward but can require the evaluation of the Hessian matrix. The final error function, $S(\mathbf{w})$, takes the form of Equation (2.13).

Using the Gaussian approximation, the hyperparameter, α , is initially chosen so as to represent our confidence in our initial assumption about the tightness of the prior density, $p(\mathbf{w})$, about zero ($\frac{1}{\alpha}$ is the variance of the prior’s Gaussian shaped distribution). The net is then trained using any standard technique which incorporates regularization, and α is treated as the regularization coefficient and adjusted every few epochs during training.

The result of this technique (as summarized by Bishop [7] and applied solely to classification) is demonstrated in Figure 2.9 and yields the following algorithm.

Bayesian Backpropagation.

1. Choose an initial positive value for the hyperparameter α . Initialize the weights in the network using values drawn from the prior distribution, $p(\mathbf{w})$.
2. Train the network using a standard non-linear optimization algorithm to minimize the total error function $S(\mathbf{w})$ given in Equation (2.13).
3. Every few cycles of the algorithm, re-estimate values for α . This can require evaluation of the Hessian matrix and evaluation of its eigenvalue spectrum, or the use of one of the approximations mentioned below.
4. Stop when criteria is met.

The evaluation of the Hessian matrix can be avoided by using an approximation to update values for α . This is simply [7]

$$\alpha^{new} = \frac{W}{2E_W}. \quad (2.20)$$

If we let ρ be the magnitude of the weight vector, $\|\mathbf{w}\|$, then

$$E_W = \frac{1}{2} \sum_{i=1}^W w_i^2 = \frac{1}{2} \rho^2 \quad (2.21)$$

and

$$\alpha^{new} = \frac{W}{\rho^2}. \quad (2.22)$$

Therefore, it would appear that changing α using this approximation simply tracks a change in ρ and allows the network to train in the vicinity of the new radial complexity. The Gaussian approximation allows Bayesian backpropagation to train the ANN using regularization with a dynamic regularization coefficient, but some would argue that it is closer to the intent of Bayes' rule to estimate the initial integration instead.

2.4.3.2 Sampling Posterior Distribution in Weight Space. One way to sidestep MacKay's Gaussian approximation is to approximate the initial integration of Equation (2.16) [7,

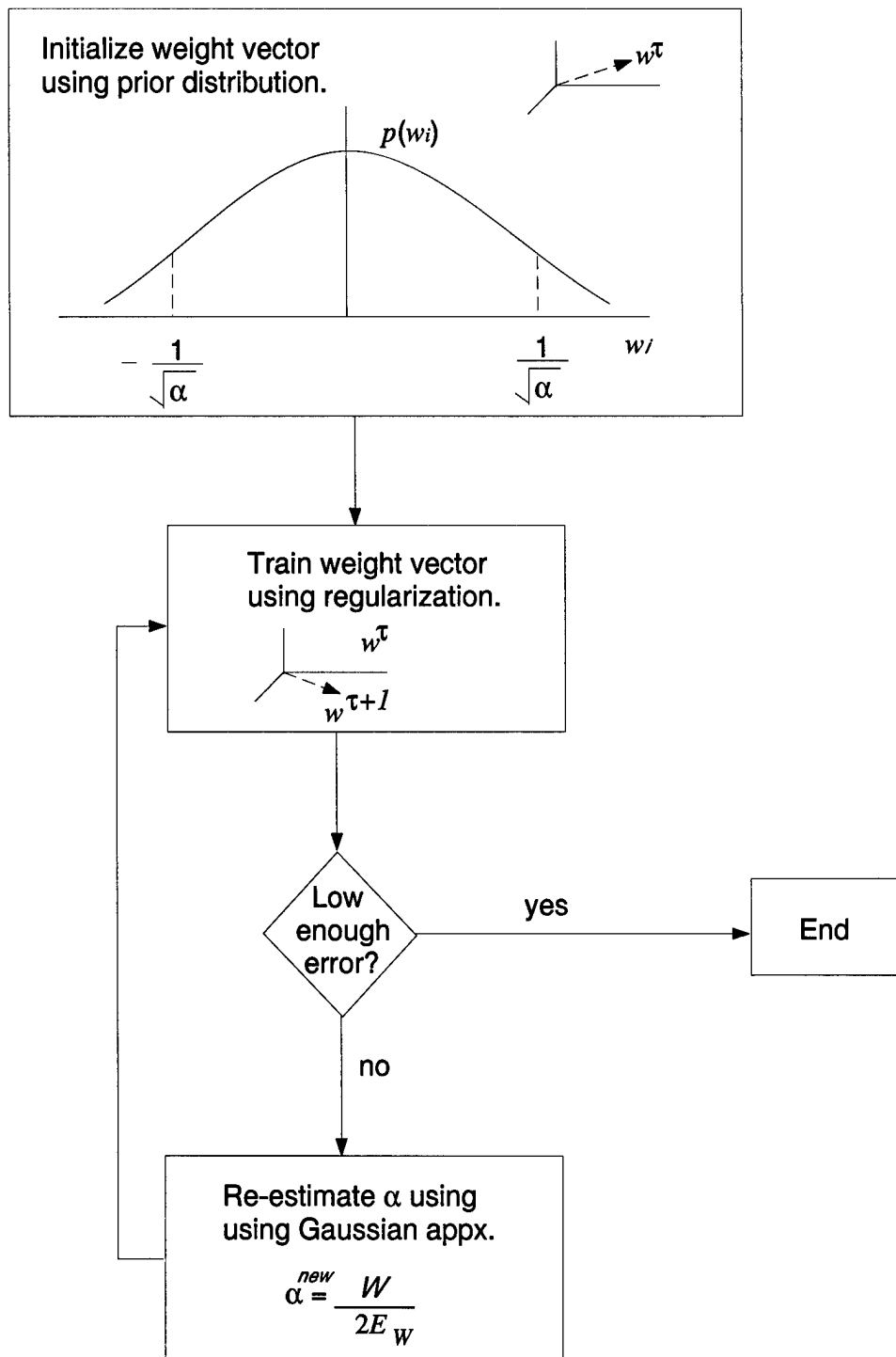


Figure 2.9 Flow chart for Bayesian training of the weights in an ANN.

46]. By finding a set of weight vectors where $p(\mathbf{w} \mid D)$ is relatively large in weight space, we can approximate the integral in Equation (2.16) with

$$\hat{y}^{(N+1)} \simeq \frac{1}{M} \sum_{i=1}^M f(\mathbf{x}^{(N+1)}, \mathbf{w}_i), \quad (2.23)$$

where M is the number of sample points in weight space. Neal [45, 46] demonstrated the applicability of this method by using a modified Monte-Carlo search to find different maximum points in the posterior distribution of the weights.

While regularization limits the magnitude of the weight vector based on the regularization coefficient, early stopping limits the magnitude of the weight vector by stopping the training (and therefore the growth of the magnitude of the weight vector) based on the error obtained on the validation set which is not used to update the weights [7].

2.4.4 Cross-Validation. Cross-validation estimates the generalization error by using the cross-validation set error as an approximation to the true generalization error. Cross-validated early stopping is a popular tool for training ANNs since it increases the generalization ability of the network after training [59]. In cross-validation, the training data set is broken into multiple sets, new training sets and validation sets. The weights are updated using any standard MLE approach, but now the error on the validation set is tracked along with the error on the training set. When cross-validated early stopping is used, the weight vector chosen is the one that minimizes the error on the cross-validation set. As Bishop points out [7], cross-validated early stopping limits the effective complexity of the ANN since the ANN is trained, not until a set error is achieved on the training set, but until the error on the validation set begins to increase. Since standard backpropagation techniques usually attempt to set the magnitude of the weight vector to a small value at initialization and the weights grow during training (see Section 3.2), the cross-validated stopping criterion is correlated with the effective complexity of the ANN rather than a training error minimization.

Figure 2.10 shows an example of cross-validation. The two curves represent errors

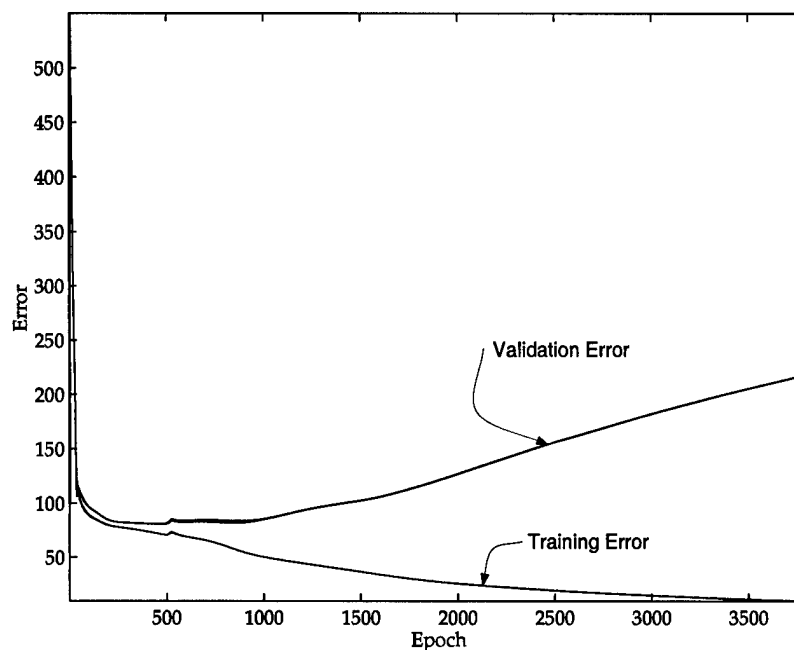


Figure 2.10 Cross-validation training on the TESSA data set. Here, the error is plotted versus the training epoch.

obtained on two data sets, training data and validation data. The training data always has the lower error since it is a biased estimate of the error that will be observed on the test data, while the validation error diverges from the training error and eventually begins to increase with increasing complexity.

2.4.5 Magnitude of the Weight Vector. The impetus behind regularization is the attempt to limit the effective complexity of the ANN by limiting the magnitude of the weight vector. Saseetharran points out that small initial weights prevent saturation of the sigmoid activation functions, but quickly grow into the saturation regions [64]. Ruck indicates that a limitation in the Bayes optimal classifier approximation would occur if the structural complexity of the network (number of hidden nodes) was too low [61]. This implies that we need some minimum complexity to approximate a Bayes optimal classifier, below which the approximation breaks down. When limiting the structural complexity of the network, a popular tool that has been used for bounding the generalization error is the “Vapnik-

Chervonenkis,” or “V-C,” dimension [77]. In the case of generalization of ANNs, we want to consider the generalization error of a given architecture for a data set of size N , denoted $g_N(y)$, as compared to the average generalization ability, $g(y)$. In that vein, we write

$$P(\max_{(y)} |g_N(y) - g(y)| > \epsilon) \leq 4\Delta(2N)e^{-\frac{\epsilon^2 N}{8}}. \quad (2.24)$$

Equation 2.24 states that the probability of the max difference in generalization errors being greater than ϵ is bounded by some function of the number of training samples, N , and ϵ . The growth function, $\Delta(N)$, gives the number of dichotomies which can be implemented by the ANN on a set of N training samples. Vapnik and Chervonenkis showed that this growth function is either identically equal to 2^N for all N , or is bounded above by the relation

$$\Delta(N) \leq N^{d_{VC}} + 1, \quad (2.25)$$

where d_{VC} is the V-C dimension, and $N^{d_{VC}}$ is the number of patterns that a given network architecture can memorize. Once the number of training samples becomes greater than $N^{d_{VC}}$, $\Delta(N)$ begins to slow down compared with the exponential term in Equation (2.24) and we can see that the right hand side of Equation (2.24) becomes arbitrarily small by making N sufficiently large. The primary downside to the V-C dimension analysis is that it yields an extremely conservative estimate of the number of training data points necessary to train an ANN to achieve good generalization results [4, 7].

Bartlett [4] showed that for ANNs used for classification, the size of the individual weights is more important for generalization than is the number of weights. He indicates that, even with a number of weights much larger than is called for based on the V-C dimension, if the *effective* complexity is limited, the generalization ability is not compromised, and, in fact, larger networks can generalize better because they can create a larger number of discriminant functions. The effective complexity based on the magnitude of the weights rather than the number of weights is what regularization and early stopping attempt to minimize.

2.5 *Summary*

In this chapter, we reviewed how it has been shown that good generalization is the key determinant when deciding on the suitability of an ANN for pattern recognition tasks. In the context of limiting the effective complexity of an ANN, the methods of regularization and cross-validated early stopping were examined and showed that limiting the magnitudes of the weights during training tends to yield good generalization results. The next chapter examines the characteristics of the radial complexity, including a method of training an ANN to an estimated radial complexity that will have improved generalization characteristics after training.

III. Achieving Good Generalization

The goal when training an ANN is to consistently achieve the lowest error on future data sets. Training the ANN to achieve low error on the training set does not guarantee low error on future data. In fact, training to low error on the training set can lead to overtraining and actually introduce higher error on the subsequent test set data. Thus, regularization pushes the weight vector in the direction of lower magnitude, thereby limiting the ability of the ANN to form overly-complex decision boundaries and overtrain, and early stopping stops the training process before overtraining can occur by also limiting the complexity of the decision boundaries.

When standard training begins in an ANN, each weight is usually initialized to some “small” random value so as to allow “growing room,” [12] while in Bayesian backpropagation, the weights are initialized based on our belief in what the final form of the weight vector will be. Since the magnitude of the weight vector is directly related to the ability of the ANN to form complex decision boundaries, the initial weight vector should be based directly on the desired initial radial complexity.

3.1 Initial Radial Complexity

As discussed in Section 2.2, the initial value of the weights plays an important and often overlooked part in the training process. One aspect of the Bayesian discussion of a prior pdf simply formalizes what has always been done when training ANNs; namely initializing the weight vector, \mathbf{w} , based on a prior belief about the final form of the weights. Given the importance of the radial complexity to the generalization ability of the final network, the expected initial radial complexity generated by our initialization of the weights must be considered before training is begun. We know that to find the expected value of the radius, $\xi(\rho)$, we have

$$\xi(\rho) = \int_0^{\infty} \rho f_P(\rho) d\rho, \quad (3.1)$$

but this requires knowledge about the probability density function, $f_P(\rho)$. Here $\xi(\cdot)$ is the expectation operator with respect to the pdf $p(\cdot)$.

3.1.1 Weights Distributed Normally. To find $f_P(\rho)$, we start with each individual weight distribution. The first case considered is when each w_i has a distribution that is $N(0, \sigma^2)$ (the prior pdf we have been discussing for Bayesian training). Define variables x_i such that x_i is $N(0, 1)$, then $w_i = \sigma x_i$ is $N(0, \sigma^2)$. Furthermore, define a variable y_i such that

$$y_i = w_i^2 = (\sigma x_i)^2 = \sigma^2 x_i^2. \quad (3.2)$$

We know that

$$\begin{aligned} \rho^2 &= y_1 + y_2 + \dots + y_W \\ &= \sigma^2 x_1^2 + \sigma^2 x_2^2 + \dots + \sigma^2 x_W^2, \end{aligned} \quad (3.3)$$

therefore

$$\frac{\rho^2}{\sigma^2} = x_1^2 + x_2^2 + \dots + x_W^2. \quad (3.4)$$

Define z such that

$$z = \frac{\rho^2}{\sigma^2}, \quad (3.5)$$

then we know that z has a Chi-square distribution, $f_Z(z)$, for the random variable Z , with W degrees of freedom [20], so

$$f_Z(z) = \begin{cases} \frac{1}{\Gamma(\frac{W}{2}) 2^{W/2}} z^{W/2-1} \exp\left(-\frac{z}{2}\right) & , z \geq 0 \\ 0 & , z < 0 \end{cases}. \quad (3.6)$$

What we want, though, is the expected value of ρ . This is found by rearranging Equation (3.5) to read

$$\rho = \sigma \sqrt{z}. \quad (3.7)$$

We can now find the expected value of ρ using

$$\begin{aligned}
\xi(\rho) &= \int_0^\infty \rho f_P(\rho) d\rho, \\
&= \xi(\sigma\sqrt{z}) \\
&= \int_{-\infty}^\infty \sigma\sqrt{z} f_Z(z) dz \\
&= \int_0^\infty \sigma\sqrt{z} \frac{1}{\Gamma\left(\frac{W}{2}\right) 2^{W/2}} z^{W/2-1} \exp\left(-\frac{z}{2}\right) dz \\
&= \frac{\sigma}{\Gamma\left(\frac{W}{2}\right) 2^{W/2}} \int_0^\infty z^{W/2-1/2} \exp\left(-\frac{z}{2}\right) dz.
\end{aligned}$$

Setting $m = \frac{W-1}{2}$ and $a = \frac{1}{2}$, and consulting the CRC Standard Mathematical Tables [6], we see that

$$\xi(\rho) = \frac{\sigma}{\Gamma\left(\frac{W}{2}\right) 2^{W/2}} \int_0^\infty z^m \exp(-az) dz \quad (3.8)$$

$$= \frac{\sigma}{\Gamma\left(\frac{W}{2}\right) 2^{W/2}} \frac{\Gamma(m+1)}{a^{m+1}}. \quad (3.9)$$

Hence,

$$\xi(\rho) = \frac{\sigma}{\Gamma\left(\frac{W}{2}\right) 2^{W/2}} \frac{\Gamma\left(\frac{W-1}{2} + 1\right)}{\left(\frac{1}{2}\right)^{(W-1)/2+1}} \quad (3.10)$$

$$= \sigma\sqrt{2} \frac{\Gamma\left(\frac{W+1}{2}\right)}{\Gamma\left(\frac{W}{2}\right)}. \quad (3.11)$$

Equation (3.11) is the exact expression for the expected value of the radius of a weight vector of length W when each element, w_i , is drawn from a prior distribution which is $N(0, \sigma^2)$. This expression becomes unwieldy for large weight vectors, though, because the calculations of the numerator and denominator are both functions of W . Since this expression becomes infeasible for even moderately large values of W , we can seek an approximation for Equation (3.11) by using Stirling's approximation for the Gamma functions in the numerator and denominator [6]. The first term in Stirling's approximation states, for $x > 10$,

$$\Gamma(x) \simeq x^x \exp(-x) \sqrt{\frac{2\pi}{x}} = x^{x-\frac{1}{2}} \exp(-x) \sqrt{2\pi},$$

which is reasonable since we are dealing with large W ($W > 20$). Using this approximation, then, yields

$$\begin{aligned} \frac{\Gamma\left(\frac{W+1}{2}\right)}{\Gamma\left(\frac{W}{2}\right)} &\simeq \frac{\left(\frac{W+1}{2}\right)^{(W+1)/2-1/2} \exp\left(-\frac{W+1}{2}\right) \sqrt{2\pi}}{\left(\frac{W}{2}\right)^{W/2-1/2} \exp\left(-\frac{W}{2}\right) \sqrt{2\pi}} \\ &= \left[\left(1 + \frac{1}{W}\right)^W\right]^{\frac{1}{2}} \left(\frac{W}{2}\right)^{\frac{1}{2}} \exp\left(-\frac{1}{2}\right). \end{aligned} \quad (3.12)$$

Observe a property of Euler's number e [6] is (since W is large)

$$\lim_{W \rightarrow \infty} \left(1 + \frac{1}{W}\right)^W = e. \quad (3.13)$$

Substituting property (3.13) into Equation (3.12) gives, for large W ,

$$\begin{aligned} \frac{\Gamma\left(\frac{W+1}{2}\right)}{\Gamma\left(\frac{W}{2}\right)} &\simeq e^{1/2} \left(\frac{W}{2}\right)^{1/2} \exp\left(-\frac{1}{2}\right) \\ &= \left(\frac{W}{2}\right)^{1/2}. \end{aligned} \quad (3.14)$$

Now, putting Equation (3.14) into Equation (3.11) gives an approximation for the expected value of ρ when W is large

$$\xi(\rho) \simeq \sigma \sqrt{W}. \quad (3.15)$$

Summarizing, if a weight vector, \mathbf{w} , of size W has elements drawn from a normal distribution such that each w_i is $N(0, \sigma^2)$, the expected value of the magnitude of that weight vector, if W is large, can be approximated using Equation (3.15).

Notice that if we look at the relationship between $\xi(\rho)$ and $\xi(\rho^2)$, we find the variance of ρ to be

$$\text{var}(\rho) = \xi(\rho^2) - [\xi(\rho)]^2. \quad (3.16)$$

Re-examining a previous variable, z , we see once again that

$$z = \frac{\rho^2}{\sigma^2}. \quad (3.17)$$

This allows us to find $\xi(\rho^2)$ since

$$\begin{aligned} \xi(z) &= \xi\left(\frac{\rho^2}{\sigma^2}\right) \\ &= \frac{1}{\sigma^2} \xi(\rho^2), \end{aligned}$$

therefore,

$$\xi(\rho^2) = \sigma^2 \xi(z). \quad (3.18)$$

This is a simple calculation since Z is a Chi-square random variable with W degrees of freedom, and the expected value for a Chi-square random variable is simply the number of degrees of freedom. This leads us to the conclusion

$$\xi(\rho^2) = \sigma^2 W. \quad (3.19)$$

Notice that in this case (remember that when W is large, $\xi(\rho) \cong \sigma\sqrt{W}$)

$$\xi(\rho^2) \cong [\xi(\rho)]^2. \quad (3.20)$$

This implies that when W is large, the variance of ρ is negligible.

3.1.2 Weights Distributed Uniformly. Next we concentrate on the problem of finding the expected value of the radius when each individual weight is drawn from a uniform distribution. In ANNs, it is common to initialize the weight vector with small values drawn uniformly between -0.5 and $+0.5$ [12]. Here, we generalize and say the weights are drawn from

a distribution that is uniform over the region $-a$ to a , ($a > 0$). This leads to a probability density function of the form

$$f_W(w) = \begin{cases} \frac{1}{2a} & \text{for } -a \leq w \leq a \\ 0 & \text{otherwise} \end{cases}. \quad (3.21)$$

We wish to find the expected value of the radius, $\rho = \sqrt{w_1^2 + w_2^2 + \dots + w_W^2}$. Define a random variable X having a pdf

$$f_X(x) = \begin{cases} \frac{1}{2} & \text{for } -1 \leq x \leq 1 \\ 0 & \text{otherwise} \end{cases}. \quad (3.22)$$

We can now state that $w_i = ax_i$ is a uniform random variable from $-a$ to a . This leads to

$$\begin{aligned} \rho &= \sqrt{w_1^2 + w_2^2 + \dots + w_W^2} \\ &= \sqrt{(ax_1)^2 + (ax_2)^2 + \dots + (ax_W)^2} \\ &= a\sqrt{x_1^2 + x_2^2 + \dots + x_W^2}, \end{aligned}$$

which yields

$$\begin{aligned} \xi(\rho) &= \xi\left(a\sqrt{x_1^2 + x_2^2 + \dots + x_W^2}\right) \\ &= a\xi\left(\sqrt{x_1^2 + x_2^2 + \dots + x_W^2}\right). \end{aligned}$$

We know that

$$\xi\left(\sqrt{x_1^2 + x_2^2 + \dots + x_W^2}\right) = \int_{-1}^1 \int_{-1}^1 \dots \int_{-1}^1 \sqrt{x_1^2 + x_2^2 + \dots + x_W^2} \times p_{x_1}(x_1)p_{x_2}(x_2)\dots p_{x_W}(x_W)dx_1dx_2\dots dx_W,$$

which leads to

$$\xi\left(\frac{\rho}{a}\right) = \left(\frac{1}{2}\right)^W \int_{-1}^1 \int_{-1}^1 \dots \int_{-1}^1 \sqrt{x_1^2 + x_2^2 + \dots + x_W^2} dx_1dx_2\dots dx_W. \quad (3.23)$$

The integration in Equation (3.23) does not have a closed form solution so we cannot give an expression for the expected value of $\frac{\rho}{a}$ as a function of W ; but we know the variance of our random variable, ρ , is

$$\text{var}(\rho) = \xi(\rho^2) - [\xi(\rho)]^2. \quad (3.24)$$

This leads to

$$\xi(\rho) = \sqrt{\xi(\rho^2) - \text{var}(\rho)}. \quad (3.25)$$

Remembering the results of our approximation when the weights were initially drawn from a normal distribution, namely that $\text{var}(\rho)$ was negligible for large W , then if we can establish that this is the case when the weights have a uniform distribution as well, we can estimate $\xi(\rho)$ using

$$\xi(\rho) \cong \sqrt{\xi(\rho^2)}. \quad (3.26)$$

We first need to calculate $\xi(\rho^2)$. We define a random variable, y , such that

$$y = w^2. \quad (3.27)$$

Using the Square Law from Thomas's book [70], we find that the pdf of y is then

$$f_Y(y) = \begin{cases} \frac{f_W(\sqrt{y}) + f_W(-\sqrt{y})}{2\sqrt{y}} & \text{for } 0 \leq \sqrt{y} \leq a \\ 0 & \text{otherwise} \end{cases}. \quad (3.28)$$

First,

$$f_W(-\sqrt{y}) = \frac{1}{2a} \quad \text{for } -a \leq -\sqrt{y} \leq 0, \quad (3.29)$$

and

$$f_W(\sqrt{y}) = \frac{1}{2a} \quad \text{for } 0 \leq \sqrt{y} \leq a. \quad (3.30)$$

But multiplying the limit terms on the negative radical by -1 yields

$$\begin{aligned} f_W(-\sqrt{y}) &= \frac{1}{2a} & \text{for } a \geq \sqrt{y} \geq 0 \\ &= \frac{1}{2a} & \text{for } 0 \leq \sqrt{y} \leq a. \end{aligned}$$

Now squaring the limits gives

$$f_W(-\sqrt{y}) = \frac{1}{2a} \quad \text{for } 0 \leq y \leq a^2, \quad (3.31)$$

and

$$f_W(\sqrt{y}) = \frac{1}{2a} \quad \text{for } 0 \leq y \leq a^2. \quad (3.32)$$

Finally, we have then

$$\begin{aligned} f_Y(y) &= \begin{cases} \frac{\frac{1}{2a} + \frac{1}{2a}}{2\sqrt{y}} & \text{for } 0 \leq y \leq a^2 \\ 0 & \text{otherwise} \end{cases} \\ &= \begin{cases} \frac{1}{2a\sqrt{y}} & \text{for } 0 \leq y \leq a^2 \\ 0 & \text{otherwise} \end{cases}. \end{aligned}$$

We will need the expected value of y ,

$$\begin{aligned} \xi(y) &= \int_{-\infty}^{\infty} y f_Y(y) dy \\ &= \int_0^{a^2} y \frac{1}{2a\sqrt{y}} dy \\ &= \frac{1}{2a} \int_0^{a^2} \sqrt{y} dy \\ &= \frac{a^2}{3}. \end{aligned}$$

Now, if we define z again to be

$$z = y_1 + y_2 + \dots + y_W, \quad (3.33)$$

then we know

$$\begin{aligned} \xi(z) &= W\xi(y_1) \\ &= W\frac{a^2}{3}. \end{aligned}$$

So then, with

$$\rho^2 = z, \quad (3.34)$$

we have

$$\xi(\rho^2) = W \frac{a^2}{3}. \quad (3.35)$$

Now, remember that we are trying to establish that $\text{var}(\rho)$ becomes negligible as W increases. Although $\xi(\rho) = a\xi\left(\sqrt{x_1^2 + x_2^2 + \dots + x_W^2}\right)$ cannot be found analytically, we can use numerical integration to establish its behavior as W increases, thereby establishing the behavior of σ_ρ as W increases. For this exercise, it will be convenient to re-express Equation (3.24) in the form

$$\begin{aligned} \frac{\sigma_\rho^2}{a^2} &= \frac{\xi(\rho^2)}{a^2} - \frac{[\xi(\rho)]^2}{a^2} \\ &= \frac{\xi(\rho^2)}{a^2} - \left[\xi\left(\frac{\rho}{a}\right)\right]^2 \\ &= \frac{W}{3} - \left[\xi\left(\sqrt{x_1^2 + x_2^2 + \dots + x_W^2}\right)\right]^2 \\ &= \frac{W}{3} - \left[\left(\frac{1}{2}\right)^W \int_{-1}^1 \int_{-1}^1 \dots \int_{-1}^1 \sqrt{x_1^2 + x_2^2 + \dots + x_W^2} dx_1 dx_2 \dots dx_W\right]^2. \end{aligned}$$

Approximating the integration in the above expression numerically for values of W from 1 to 10, we can glean the behavior of the variance of the complexity as W increases. Figure 3.1 shows how the variance of the radial complexity decreases with increasing W and Figure 3.2 demonstrates how the expected value of the square of the complexity grows along with the square of the expected value of the complexity with increasing W . Figure 3.3 shows the difference between the calculated approximation of $\xi(\rho)$ and the average magnitude of ρ observed when initializing a set of weights using a uniform distribution and this analysis supports our use of the approximation in Equation (3.36).

From this analysis, it is reasonable to use the approximation of Equation (3.26) and estimate the expected value of the complexity as

$$\xi(\rho) \cong a\sqrt{\frac{W}{3}}. \quad (3.36)$$

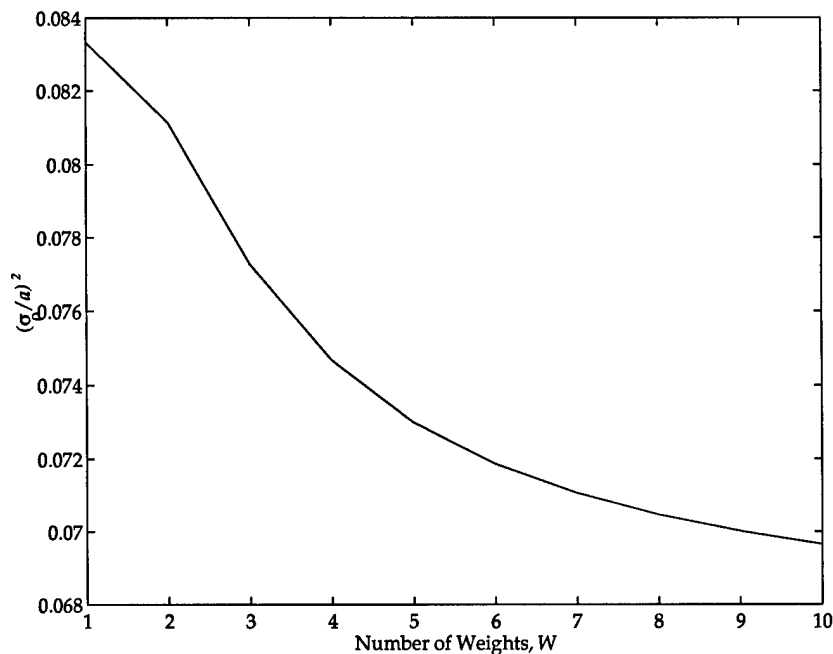


Figure 3.1 Demonstration of the decay of the variance of the radial complexity, $(\frac{\sigma_e}{a})^2$, with increasing W .

With the determination of the expected initial radial complexity, what is the behavior of that radial complexity during different types of weight training?

3.2 Consistent Behavior of Radial Complexity During Training

One method of determining when the weights in the ANN are “good enough” is based on the training error; once the training set error is at or below some specified value, one can discontinue training and test the ANN’s ability to classify a test data set [7,59]. Having shown that the distribution used to initialize the weights determines the expected value of the initial radial complexity of the weight vector, it is desired to establish the relationship between the radial complexity and the training error during training of the ANN since this radial complexity contributes greatly to the ability of the ANN to generalize well to new data. In this section, we demonstrate one aspect of the behavior exhibited by the radial complexity during training of an ANN using standard backpropagation and regularization with sum-squared and softmax error on the OCR and TESSA data sets. Using an ANN with

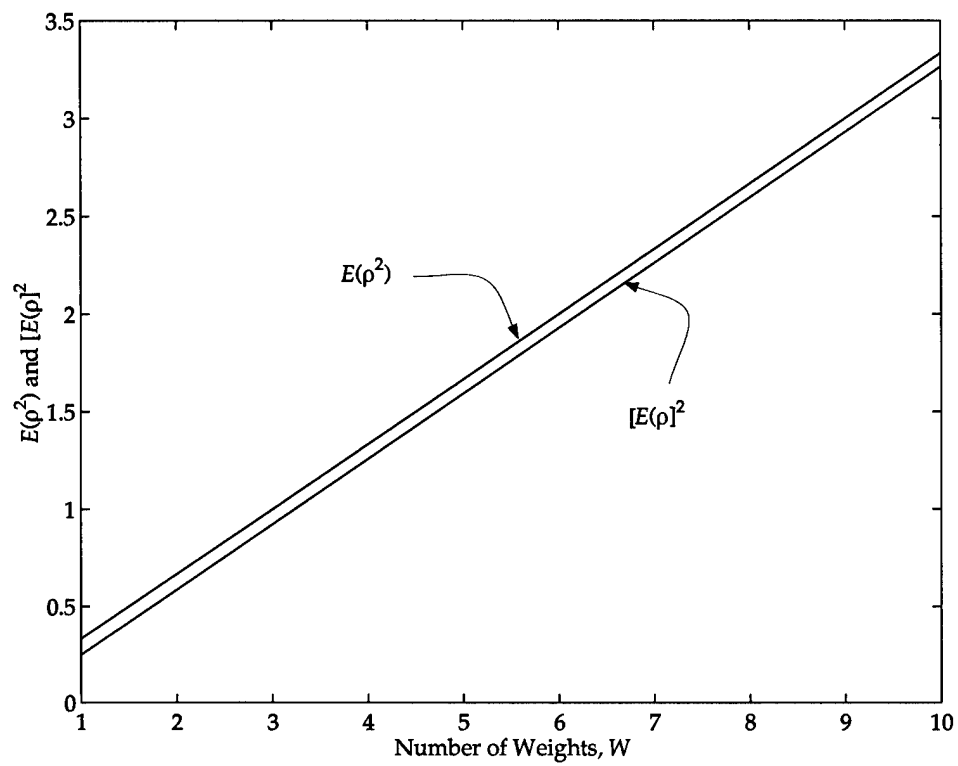


Figure 3.2 Demonstration of the growth of the expected value of the radial complexity squared and the square of the expected value of the radial complexity with increasing W .

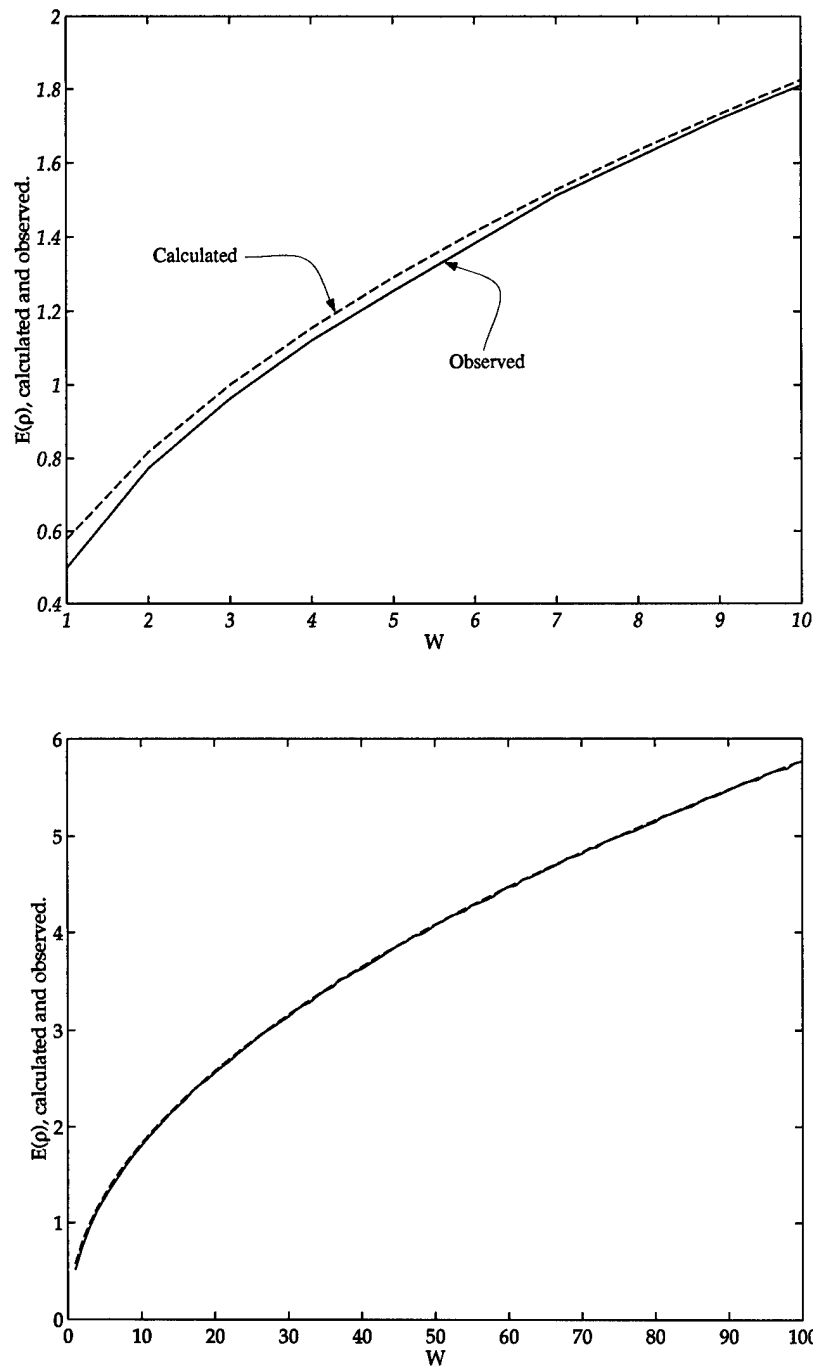


Figure 3.3 Demonstration of the growth of the approximate expected value of the radial complexity (dotted) and the average observed magnitude of the radial complexity (solid) with increasing W . The plot on the top shows that, for small W , the approximation is not very good; but the plot on the bottom shows that for increasingly large W , the approximation is much better.

70 hidden nodes and 10 output nodes (one for each class), the demonstrations in this section show an almost deterministic behavior of the radial complexity as the training occurs. In Figures 3.4, 3.5, 3.6 and 3.7, looking at the plots in each figure going from left to right, the weights are re-initialized using values drawn from a prior Gaussian distribution which is set for low radial complexity ($\xi(\rho) = .1$) for standard backpropagation or high complexity ($\xi(\rho) = 95$) for Bayesian backpropagation. Each figure consists of 5 different runs from left to right. The only difference between each plot is that the weights are re-initialized using the same distribution each time. The complexity measure is that of radial complexity, or the magnitude of the weight vector. Figure 3.4 shows the consistency of radius growth when using SSE and standard backprop on the OCR data set, Figure 3.5 shows the consistency of radius decay when using SSE and Bayesian backprop on the OCR data set, Figure 3.6 shows the consistency of radius growth when using softmax error and vanilla backprop on the OCR data set, and Figure 3.7 shows the consistency of radius decay when using softmax error and Bayesian backprop on the OCR data set. Notice that as the training ensues, even though the weights have been *randomly* reinitialized each time, the way the error and radial complexity change is consistent; that is to say that the error and radial complexity change in the same manner with each subsequent run.

This consistent behavior leads us to the conclusion that, when basing the stopping criteria on the training set error, the final radial complexity of the ANN is the same to within the variance of the initial magnitude of the weight vector; but is the training method responsible for the consistent behavior of the radial complexity? The next section demonstrates the results of using a non-backpropagation training method for the weights.

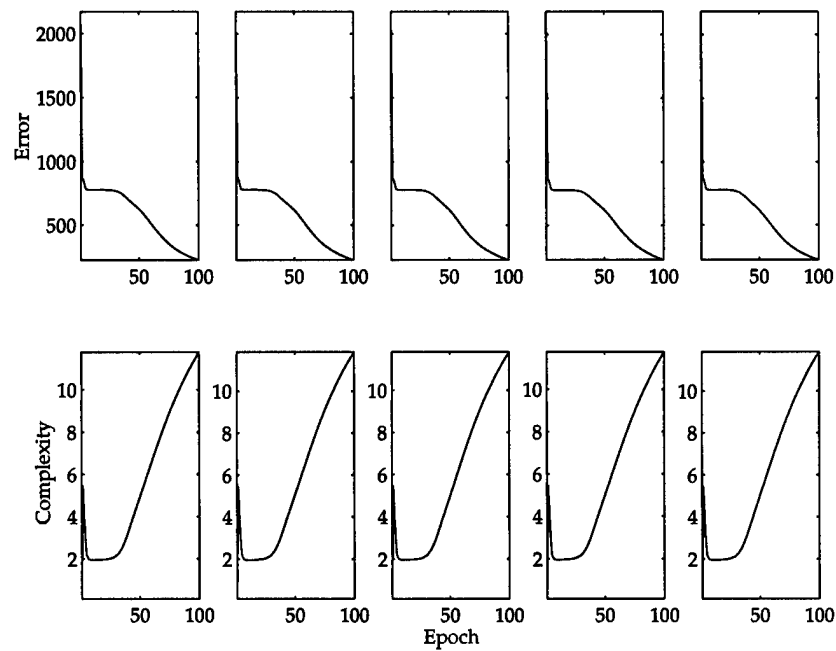


Figure 3.4 Each plot along the top row tracks the training set error versus the training epoch while each plot along the bottom row demonstrates how the magnitude of the weight vector changes as training ensues. Each column is an independent training run with the only change between columns being the re-initialization of the weight vector.

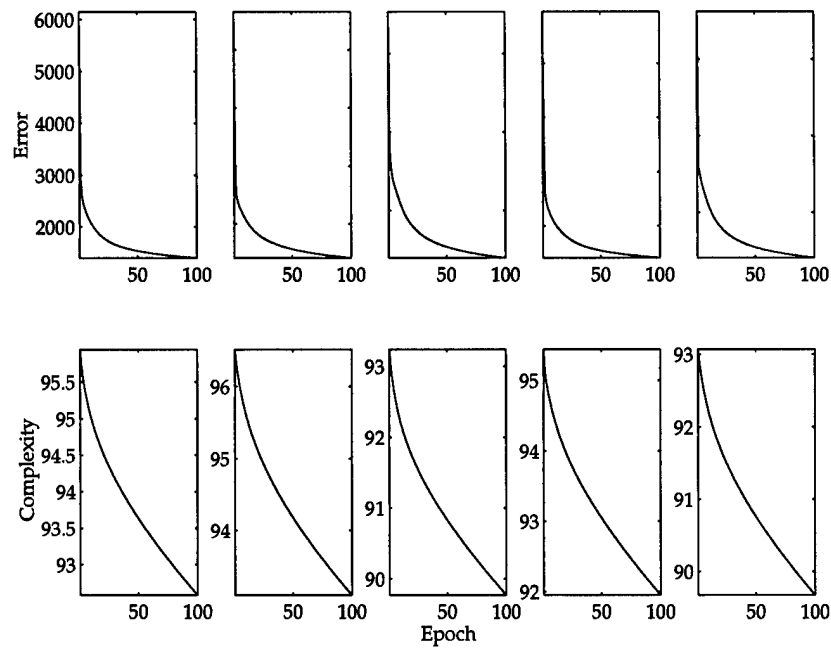


Figure 3.5 Each plot along the top row tracks the training set error versus the training epoch while each plot along the bottom row demonstrates how the magnitude of the weight vector changes as training ensues. Each column is an independent training run with the only change between columns being the re-initialization of the weight vector.

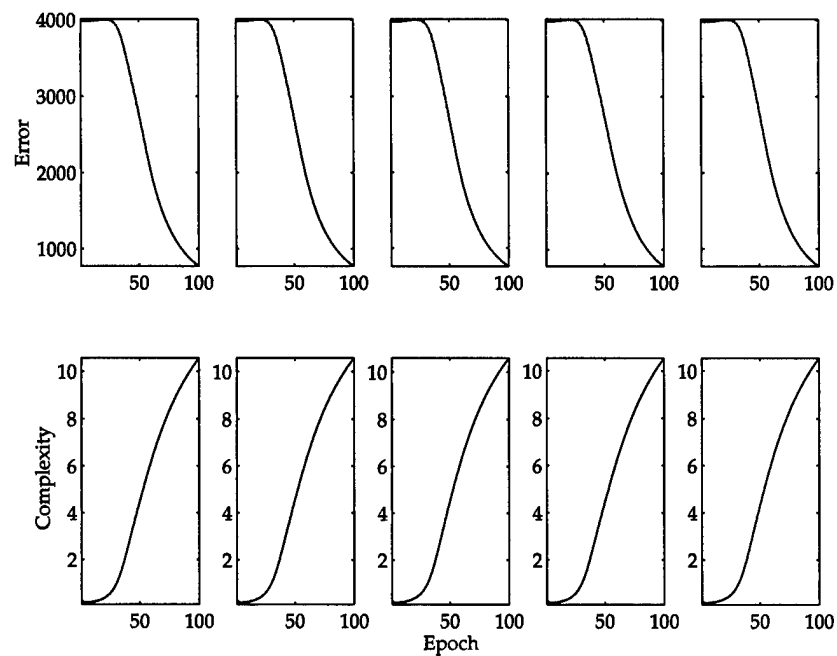


Figure 3.6 Each plot along the top row tracks the training set error versus the training epoch while each plot along the bottom row demonstrates how the magnitude of the weight vector changes as training ensues. Each column is an independent training run with the only change between columns being the re-initialization of the weight vector.

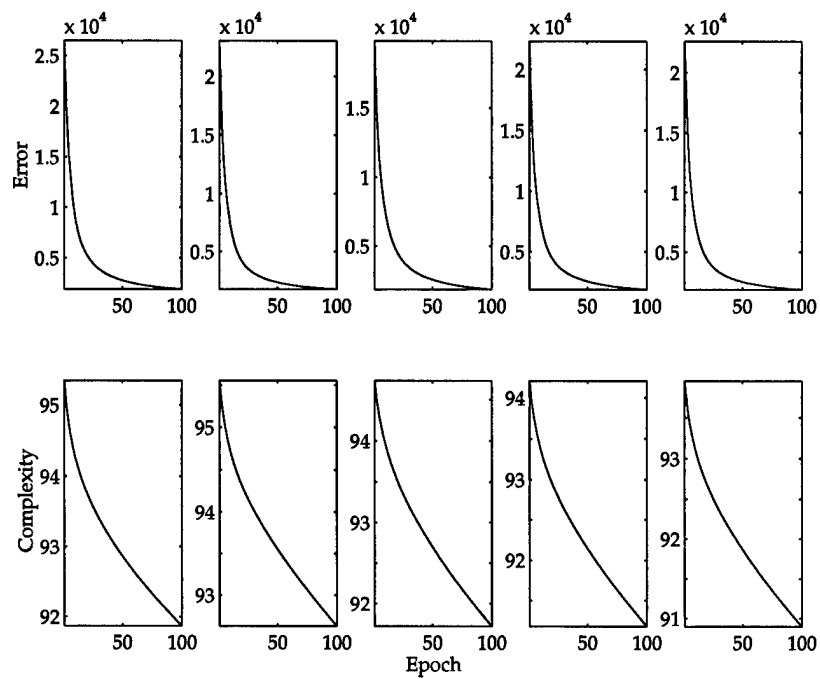


Figure 3.7 Each plot along the top row tracks the training set error versus the training epoch while each plot along the bottom row demonstrates how the magnitude of the weight vector changes as training ensues. Each column is an independent training run with the only change between columns being the re-initialization of the weight vector.

3.2.1 Consistency of Training Behavior When Using EP Training. The consistency demonstrated above is *not* a result of using backpropagation. Backpropagation is not the only method for setting the weights in an ANN. In fact, the final form the weight vector takes should be independent of the method used to obtain that vector. Evolution programs provide an alternative method for ANN weight training. The EP technique discussed in section 2.3.1.3 was used to analyze the behavior of the radius as shown in Figure 3.8. This

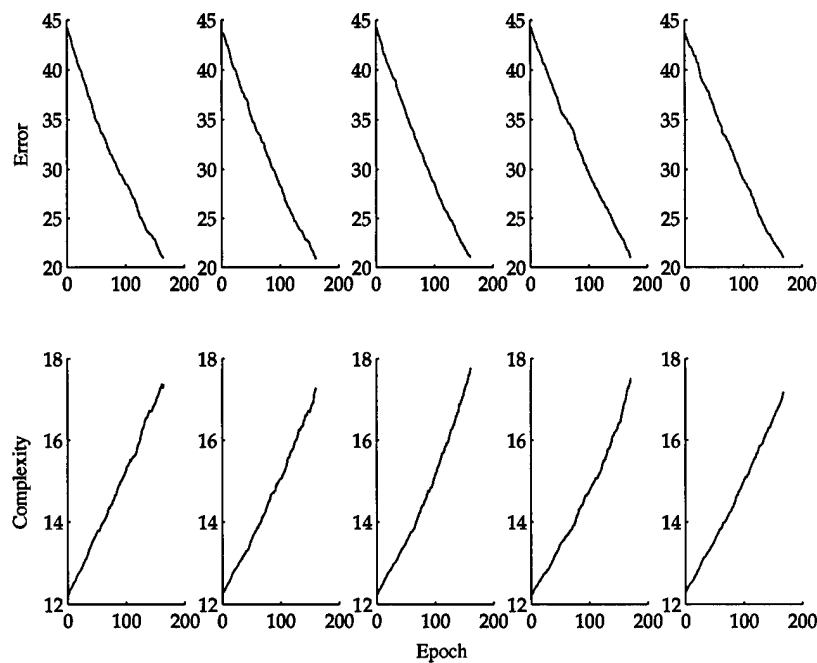


Figure 3.8 Each plot along the top row tracks the training set error versus the training epoch while each plot along the bottom row demonstrates how the magnitude of the weight vector changes as training ensues. Each column is an independent training run with the only change between columns being the re-initialization of the weight vector.

figure demonstrates how the radial complexity behaves when using an evolution program. The squashing function is incorporated into the EP to preclude getting the same solution. The radial complexity also displays consistent behavior when being trained using the EP. Clearly, the behavior of the radial complexity as the ANN training seeks out a weight vector yielding a minimum error is not dependent on the backpropagation of error routine.

The above analysis was carried out on a hand-written character set. Does this data set give rise to an error surface that would cause the radial complexity to exhibit this consistent behavior? The next section shows that different data also gives rise to this type of behavior.

3.2.2 Consistency of Training Behavior When Training on TESSA Data Set .

Would the radius still exhibit the same consistent behavior on a different data set than the character data set in the previous section? The TESSA data set contains infrared images that are very hard to classify. Training on this data set reveals that the radial complexity behaves consistently during training even when the network is not converging to a solution (Figure 3.9). This figure demonstrates the consistent behavior of the radius

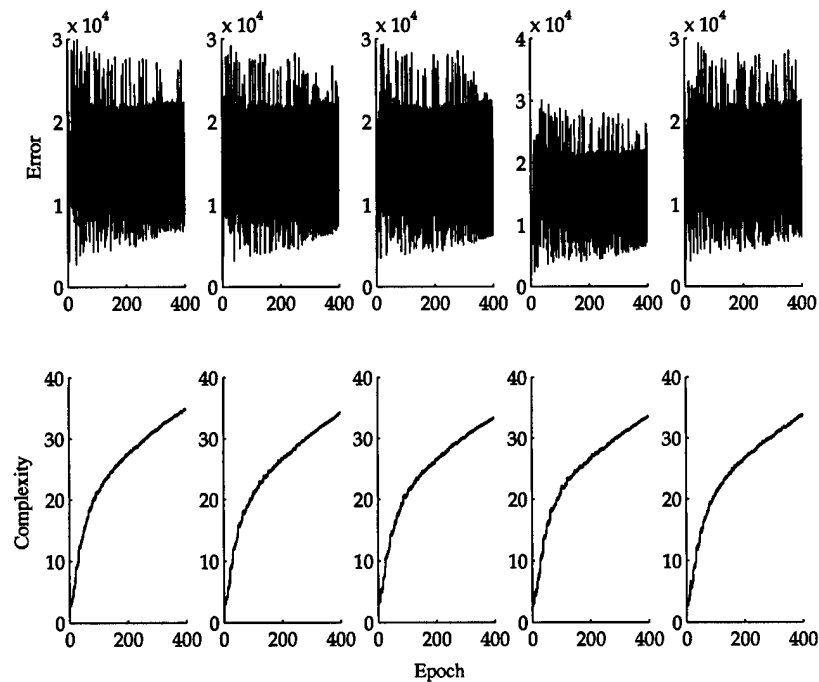


Figure 3.9 Each plot along the top row tracks the training set error versus the training epoch while each plot along the bottom row demonstrates how the magnitude of the weight vector changes as training ensues. Each column is an independent training run with the only change between columns being the re-initialization of the weight vector.

when training on the infrared target data set. Note the consistency with which the radius behaves even when the error is not converging to a solution. Once again, we see that the behavior of the radial complexity remains consistent between runs even when the weights are re-initialized to different random values each time.

This analysis of the behavior of the radial complexity during training also gives us better insight as to how cross-validated early stopping limits the magnitude of the weight

vector; the training (and therefore the growth of the radial complexity) is stopped earlier than if training set error were the stopping criterion.

The consistency of behavior of the radial complexity is *independent* of the training method and the data set. Having shown in this section that re-initializing an ANN with a different set of weights drawn from the same distribution does not guarantee a different radial complexity when training is complete (in fact, training to some set training error will most likely put us into the same radial complexity every time), and that the radial complexity consistently changes during training, we need a way to estimate at what radial complexity to halt the training so that we can achieve good generalization after the training is complete. The next section presents the steps taken to obtain a good generalization radial complexity and then train to that complexity. Cross-validation plays a key role in empirically establishing the relationship between generalization error and radial complexity.

3.3 Cross-Validation Radial Complexity Estimation

When using the full data set as the training set, we cannot use cross-validated early stopping since we now have no data for a cross-validation set. We can, though, estimate the expected radial complexity for the full data set that will provide enough constraint on the discriminant boundaries such that a Bayes optimal discriminant function is best approximated. White [83] describes how the structural complexity of an ANN grows as the number of training data points, or experience, grows. Bartlett [4] augments this analysis with his proof that the magnitude of the weights (quantified here as the radial complexity) can be more important to generalization than the number of weights. Therefore, in classification problems for which ANNs are well suited, the effective complexity should also grow in the same manner as the structural complexity when the training data set grows [7,59]. Estimating the radial complexity allowed for a data set of size N that provides good generalization characteristics is accomplished by using cross-validation on smaller data sets, slowly adding more data and each time using cross-validation; this allows us to see how the radial complexity that achieves good generalization grows as the size of the data set increases. Using regression on these radial complexities (letting the radial complexity be a function of M , the

number of training points), we can infer the final expected radial complexity of the ANN that will provide good generalization characteristics when training using the entire data set, and use hyperspherical backpropagation to train at that radial complexity. This allows the use of all data to train the ANN and still expect good generalization characteristics when training is complete.

3.3.1 Generalization Error. Let the training set be,

$$D_{train} = \{(\mathbf{x}^{(1)}, \mathbf{t}^{(1)}), \dots, (\mathbf{x}^{(N)}, \mathbf{t}^{(N)})\}, \quad (3.37)$$

and the loss function be $l(D_{train}, \mathbf{w}^\tau)$. The error function on the training data set is,

$$E_{train}(\mathbf{w}) = \frac{1}{N} \sum_{n=1}^N l(D_{train}, \mathbf{w}). \quad (3.38)$$

Using any learning rule (e.g., backpropagation), one gets a sequence of weight vectors, $\{\mathbf{w}^\tau\}$, where τ is the time step taken by one pass through the training data (one epoch). The loss function is typically either the least square error

$$l(D_{train}, \mathbf{w}^\tau) = \frac{1}{2} \sum_{k=1}^K \left| t_k^{(n)} - y_k(\mathbf{x}^{(n)}, \mathbf{w}^\tau) \right|^2,$$

or the softmax error

$$l(D_{train}, \mathbf{w}^\tau) = - \sum_{k=1}^K t_k^{(n)} \ln y_k(\mathbf{x}^{(n)}, \mathbf{w}^\tau).$$

The generalization error for a given weight vector, \mathbf{w} , is the expected value, $\xi[\cdot]$, of the loss function over all possible future example test data sets,

$$E_{gen}(\mathbf{w}) = \xi_{Test}[l(D_{test}, \mathbf{w})]. \quad (3.39)$$

Optimally, the trained network will yield the minimum generalization error,

$$E_{gen}^* = \min_{\mathbf{w}} [E_{gen}(\mathbf{w})],$$

and is therefore defined as

$$E_{gen}^* \equiv \min_{\mathbf{w}} [\xi_{Test} [l(D_{test}, \mathbf{w})]] .$$

Given a cross-validation data set,

$$D_{cv} = \{(\mathbf{x}_{cv}^{(1)}, \mathbf{t}_{cv}^{(1)}), \dots, (\mathbf{x}_{cv}^{(N)}, \mathbf{t}_{cv}^{(N)})\}, \quad (3.40)$$

the error on the cross-validation set is,

$$E_{cv}(\mathbf{w}) = \frac{1}{N} \sum_{n=1}^N l(D_{cv}, \mathbf{w}) .$$

The generalization error can be approximated as

$$E_{gen}(\mathbf{w}^\tau) \simeq E_{cv}(\mathbf{w}^\tau) ,$$

where τ is the time step, or epoch. If cross-validated early stopping is used, then the optimal generalization error can be approximated as,

$$E_{gen}^* \simeq \min_{\tau} [E_{cv}(\mathbf{w}^\tau)] .$$

The ability of the network to form complex discriminant boundaries is quantified using the radial complexity, ρ , therefore we are trying to empirically establish the relationship between the generalization error and the radial complexity. Given,

$$\begin{aligned} E_{gen}(\mathbf{w}) &= f\left(\sqrt{w_1^2 + w_2^2 + \dots + w_W^2}\right) \\ &= E_{gen}(\|\mathbf{w}\|) , \end{aligned}$$

then the generalization error can be expressed as a function of ρ such that $\rho = \|\mathbf{w}^{\tau^*}\|$, where τ^* is the integer where training is stopped. The approximate generalization error can then

be represented as,

$$E_{gen}(\|\mathbf{w}^{\tau^*}\|) \simeq E_{cv}(\mathbf{w}^{\tau^*}),$$

therefore

$$E_{gen}(\rho) \simeq E_{cv}(\mathbf{w}^{\tau^*}). \quad (3.41)$$

If several training and cross-validation sets are available, the expected optimal generalization error as a function of ρ is then estimated by taking the average minimum error over all cross-validation sets

$$E_{gen}^*(\rho) \simeq \frac{1}{Q} \sum_{q=1}^Q \left[\min_{\rho} [E_{cv}^{(q)}(\rho)] \right], \quad (3.42)$$

where $E_{cv}^{(q)}$ is the error on the cross-validation set at the q^{th} run, and Q is the number of separate cross-validation runs. Note that the dependency of the approximate optimal generalization error on the radial complexity is now explicit since individual solution weight vectors are not the desired outcome, but instead the relationship between generalization error and the magnitude of the weight vector is of primary interest.

Define the optimal radial complexity, $\rho_q^{(t)}$, for a data set of size M_t at run q as the one that corresponds to the weight vector that yields minimum error on the validation set for that run. The optimal radial complexity, $\rho_{OPT}^{(t)}$, for a classification problem represented by subsets of size M_t can then be defined as the average of the radial complexities indicated by cross-validated early stopping on those data sets,

$$\rho_{OPT}^{(t)} = \frac{1}{Q} \sum_{q=1}^Q \rho_q^{(t)}. \quad (3.43)$$

This average radial complexity is the one that yields the average minimum cross-validation error over the Q cross-validation runs. The minimum radial complexity is not what is sought since the radial complexity indicated by each cross-validation run will be dependent on how closely matched the randomly chosen training set is to the randomly chosen validation set.

3.3.2 Early Stopping at the Estimated Radial Complexity. As the number of training data points grows, the expected good-generalization radial complexity grows as well. We

trained an ANN so as to obtain 15 different radial complexities. The radial complexities demonstrated increasing growth with increasing number of training data points. Each radial complexity was an average over 100 runs whose training and validation data was randomly drawn from the larger data set. Figure 3.10 shows the results of using cross-validation on data sets with 5 training data points from each class, while Figure 3.11 shows the results of

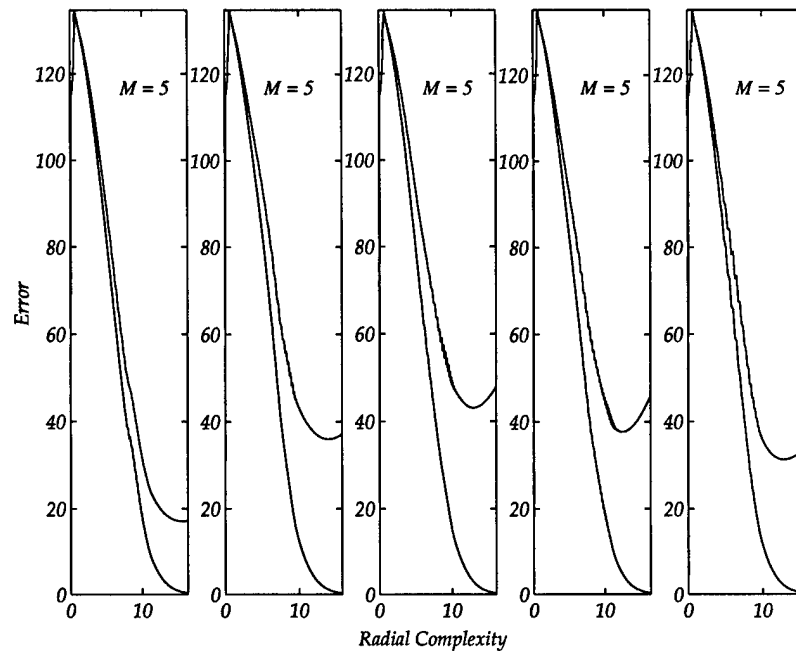


Figure 3.10 Cross-Validation result for 5 training data points from each class.

using cross-validation on data sets with 50 training data points from each class. Once again, the plots in each figure show the error versus the radial complexity. The radial complexity for good generalization in each plot is that which leads to minimum error on the validation error curve. The difference between each plot in a given figure is re-initialization of the weights and choosing a random data set for training/validation. Looking at these two figures, we see that the radial complexity yielding good generalization (where the upper validation error curve is minimum) grows as we increase the number of training data points. Training was done for 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, and 75 data points from each class.

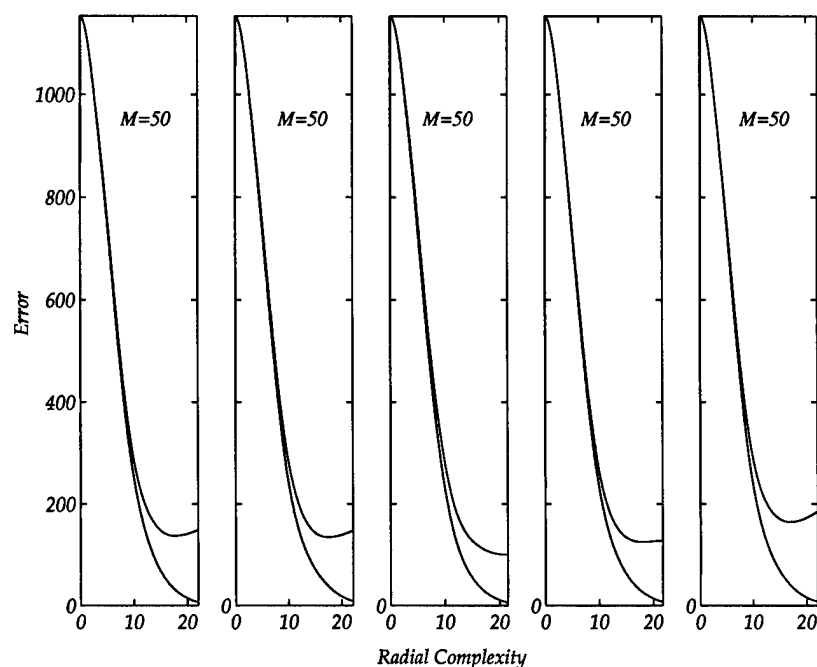


Figure 3.11 Cross-Validation result for 50 training data points from each class.

Figure 3.12 shows the average good generalization radial complexity calculated for each size data set. Using regression on this data, we can estimate the radial complexity for a large number of training data points which yields good generalization. The radial complexity estimated for 100 data points that yields good generalization is on the order of 22. Looking at Figure 3.13, notice that a radial complexity of 22 is indeed representative of the region in which the error on the validation set begins to increase.

We randomly generated a small data set (100 vectors from each class) from the TESSA data (3 class) to be used as the training data set and constructed an ANN with 70 hidden nodes. Using the technique described in the previous section, we used cross-validated early stopping with half the training data as validation data and half remaining as training data. The generalization error was then estimated as the error obtained on the remaining data (containing about 10,000 points) as per Equation 3.42. The radial complexity expected to provide good generalization for the full data set was estimated, and the small data set was trained until the radial complexity reached that value. The effects of overtraining can be seen in Figure 3.14. Overtraining yields a percentage of correctly classified test points of

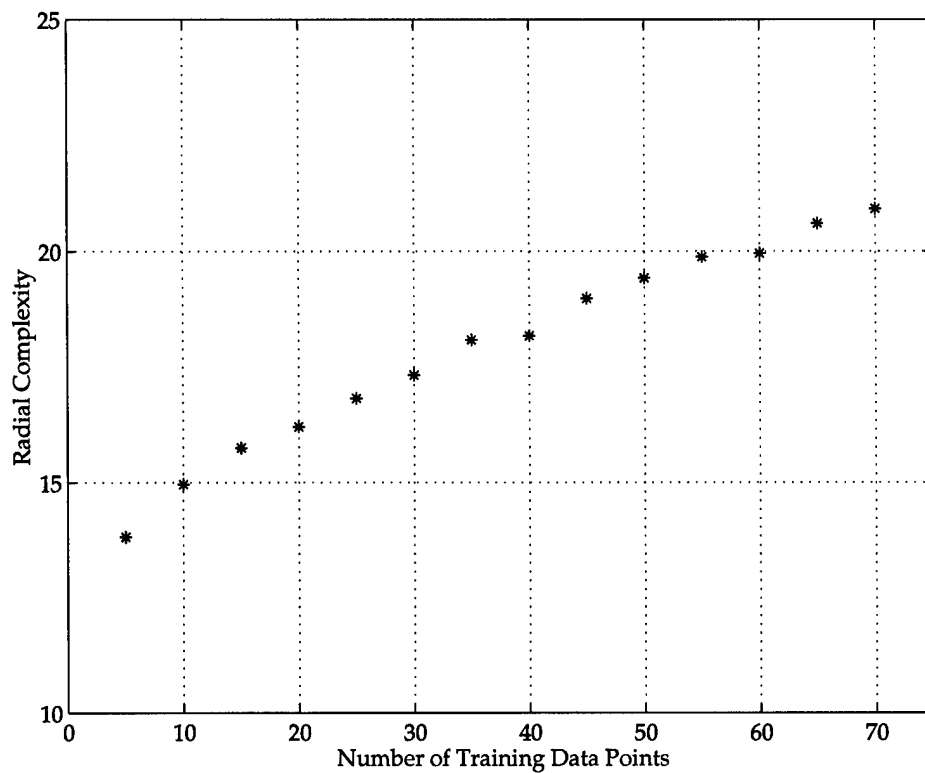


Figure 3.12 Growth of the radial complexity providing good generalization as a function of the number of training data points.

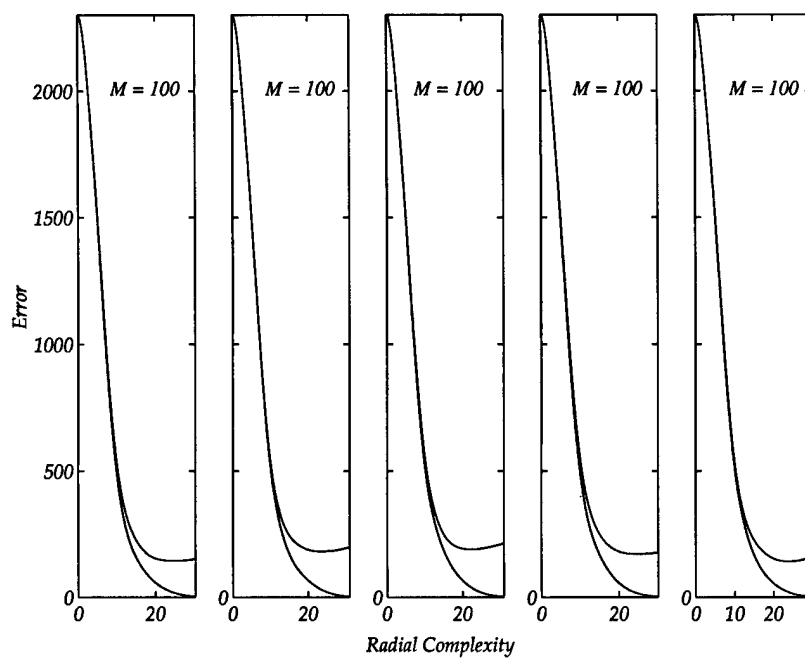


Figure 3.13 Cross-Validation result for 100 training data points from each class.

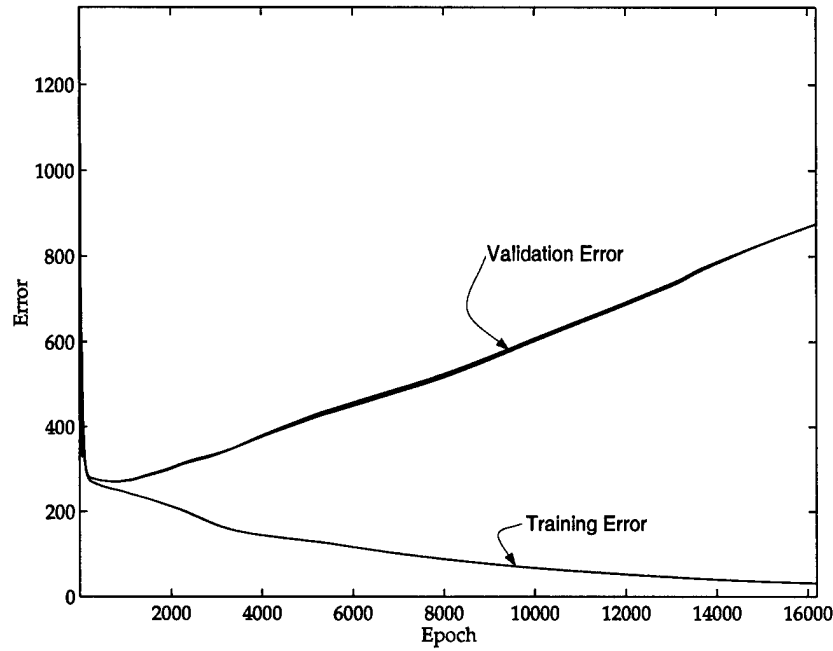


Figure 3.14 Using cross-validation, we can see that the error on the validation set grows rapidly even as the training error decreases.

51.75%. While cross-validated early stopping improves the correct classification rate to 61.12%, the best classification rate of 63.28% is achieved when using the full data set and stopping training at the estimated good generalization radial complexity, ρ_N . These results are summarized in Table 3.44.

Stopping based on:	E_{train}	E_{cv}	ρ_N
Percent correctly classified	51.75%	61.12%	62.75%

(3.44)

Each result is an average classification rate over 10 runs. The observed classification accuracy when using the full data set improves 3% over that observed when using a smaller part for true cross-validated early stopping, and over 14% over that observed when stopping training when a low error on the training data is reached.

We have already shown how standard training of the ANN leads to a consistent change in the radial complexity during training of the weights, so moving the coordinate system in which the weights are represented from Cartesian to hyperspherical allows us to lock in

the radial complexity while changing the other parameters. By converting our coordinate system from Cartesian to hyperspherical (outlined in Appendix B), we can influence the parameters of the weight vector not associated with the radial complexity (magnitude of the weight vector).

3.4 Training at a Fixed Radial Complexity

Now that we have estimated at which radial complexity to train our network to obtain good generalization, what would the effect be of restricting training to that radial complexity? This analysis can be carried out by using hyperspherical methods such as an Evolution Program or hyperspherical backpropagation which operate on the angles only. Since we have defined our effective complexity in terms that are complementary to the hyperspherical coordinate transformation, hyperspherical methods such as these provide an ideal way to confine a weight vector to a specific radial complexity during training.

3.4.1 Genetic Approach. Using EPs we can generate a population of weight vectors, convert them to hyperspherical coordinates, use evolution on the angles, and obtain a solution that maintains a constant radial complexity. Figure 3.15 shows the results of using an EP to train the ANN at a given radial complexity to classify the hand-written character set. The

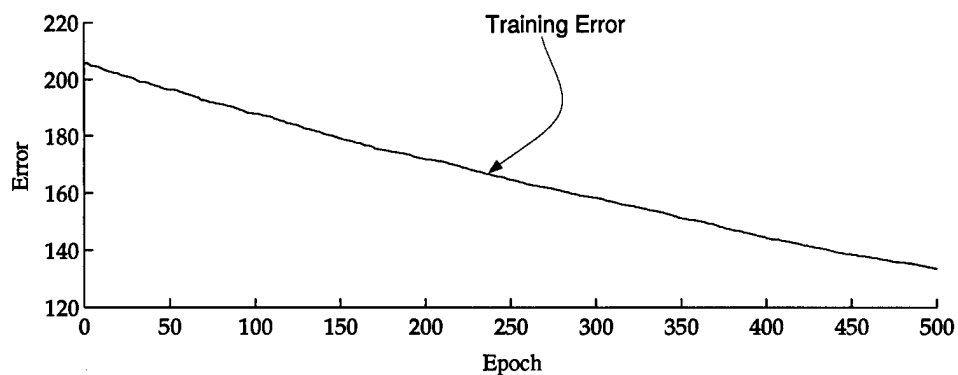


Figure 3.15 Using an EP to train the ANN at a specific radial complexity.

error used is the same error used with backprop, but the weight angles are updated using genetic methods rather than backpropagation methods. Each training epoch here is not just a pass through the training data, but a pass that yielded a lower error than the previous

iteration. This method demonstrates the tendency of the training error to decrease, but is very slow compared to hyperspherical backpropagation which constrains the weight updates to travel in the direction of a lower error.

3.4.2 Hyperspherical Backpropagation Approach. Dunne first suggested transforming the weight vector using polar coordinates (since he was working with only 2 weights) [17] to analyze the behavior of the weights over time. That research was limited to a very small dimensional weight space; here we allow for any number of weights since real-world problems usually demand a very large network to reach a solution. Thus, we use the term “hyperspherical” coordinates to denote any radial coordinate system that has more than three Cartesian coordinates as its starting frame of reference. Given that there is some ideal hypershell in which lies the weight vector that provides the best generalization, then one hopes this weight vector radius is what the training methods limiting weight values seeks; once we determine this value, we should confine the training of the weight vector to that radial complexity so as not to harm the generalization characteristics of the trained ANN. Here, we refer to any method that updates the weights using a combination of standard backpropagation and hyperspherical coordinates as hyperspherical backpropagation. If we desire to update the angles directly, then we need to know the effects of changing those angles in hyperspherical coordinates, i.e., what is the change in the error due to a change in a given weight vector angle?

For hyperspherical backpropagation, we have

$$\theta^{\tau+1} = \theta^{\tau} - \eta \frac{\partial E_D(\theta^{\tau})}{\partial \theta}. \quad (3.45)$$

For notational convenience, we will arrange the angles in the same manner that we arrange the weights. Let the weights be expressed as

$$\begin{aligned} W1 &= \begin{bmatrix} W1^1 & W1^2 & \dots & W1^R \end{bmatrix}, \\ W2 &= \begin{bmatrix} W2^1 & W2^2 & \dots & W2^{S1} \end{bmatrix}, \end{aligned}$$

where

$$W1^i = \begin{bmatrix} W1_{1,i} \\ W1_{2,i} \\ \vdots \\ W1_{S1,i} \end{bmatrix} \quad i = 1, 2, \dots, R ,$$

and

$$W2^j = \begin{bmatrix} W1_{1,j} \\ W1_{2,j} \\ \vdots \\ W1_{K,j} \end{bmatrix} \quad j = 1, 2, \dots, S1 .$$

Let a concatenated (column) weight vector, \mathbf{w} , be

$$\begin{aligned} \mathbf{w} &= \left[(W1^1)^T \ (W1^2)^T \ \dots (W1^R)^T \ (B1)^T \ (W2^1)^T \ (W2^2)^T \ \dots (W2^{S1})^T \ (B2)^T \right]^T \\ &= [w_1 \ w_2 \ \dots \ w_W]^T . \end{aligned}$$

Now the coordinate transformation from Appendix B can be used to yield a vector of angles and a magnitude such that

$$\begin{aligned} \mathbf{w} &= [\theta_1, \theta_2, \dots, \theta_{W-1}, \rho]^T \\ &= \left[(\theta^{W1^1})^T \ (\theta^{W1^2})^T \ \dots (\theta^{W1^R})^T \ (\theta^{B1})^T \ (\theta^{W2^1})^T \ (\theta^{W2^2})^T \ \dots (\theta^{W2^{S1}})^T \ (\theta^{B2})^T \right]^T \end{aligned}$$

$$\begin{aligned} \theta^{W1} &= \begin{bmatrix} \theta^{W1^1} & \theta^{W1^2} & \dots & \theta^{W1^R} \end{bmatrix} , \\ \theta^{W2} &= \begin{bmatrix} \theta^{W2^1} & \theta^{W2^2} & \dots & \theta^{W2^{S1}} \end{bmatrix} , \end{aligned}$$

$$\begin{aligned}
\theta^{W1} &= \begin{bmatrix} \theta_{1,1}^{W1} & \theta_{1,2}^{W1} & \cdots & \theta_{1,R}^{W1} \\ \theta_{2,1}^{W1} & \theta_{2,2}^{W1} & & \theta_{2,R}^{W1} \\ \vdots & & \ddots & \\ \theta_{S1,1}^{W1} & \theta_{S1,2}^{W1} & & \theta_{S1,R}^{W1} \end{bmatrix}, \\
\theta^{B1} &= \begin{bmatrix} \theta_1^{B1} \\ \theta_2^{B1} \\ \vdots \\ \theta_{S1}^{B1} \end{bmatrix}, \\
\theta^{W2} &= \begin{bmatrix} \theta_{1,1}^{W2} & \theta_{1,2}^{W2} & \cdots & \theta_{1,S1}^{W2} \\ \theta_{2,1}^{W2} & \theta_{2,2}^{W2} & & \theta_{2,S1}^{W2} \\ \vdots & & \ddots & \\ \theta_{K,1}^{W2} & \theta_{K,2}^{W2} & & \theta_{K,S1}^{W2} \end{bmatrix}, \\
\theta^{B2} &= \begin{bmatrix} \theta_1^{B2} \\ \theta_2^{B2} \\ \vdots \\ \theta_{K-1}^{B2} \end{bmatrix}.
\end{aligned}$$

Accordingly, we assign the angles to a given weight as

$$W1_{j,i} = \rho \sin \theta_1 \sin \theta_2 \dots \sin \theta_{j,i-1}^{W1} \cos \theta_{j,i}^{W1}, \quad (3.46)$$

where the subscript $j, i - 1$ simply implies the angle preceding the ji angle in the W -dimensional weight space. Similarly,

$$B1_j = \rho \sin \theta_1 \sin \theta_2 \dots \sin \theta_{j-1}^{B1} \cos \theta_j^{B1}, \quad (3.47)$$

$$W2_{k,j} = \rho \sin \theta_1 \sin \theta_2 \dots \sin \theta_{k,j-1}^{W2} \cos \theta_{k,j}^{W2}, \quad (3.48)$$

and

$$B2_k = \rho \sin \theta_1 \sin \theta_2 \dots \sin \theta_{k-1}^{B2} \cos \theta_k^{B2}. \quad (3.49)$$

The angle updates are derived in Appendix B as

$$\begin{aligned} \theta_{k,j}^{W2}(\tau+1) &= \theta_{k,j}^{W2}(\tau) - \eta \sum_{n=1}^N (y_k^{(n)} - t_k^{(n)}) \times \\ &\quad (-z_j w_{kj} \tan \theta_{k,j}^{W2}(\tau) + z_{(j+1)} W2_{k(j+1)} \cot \theta_{k,j}^{W2}(\tau) + \dots \\ &\quad + z_{S1} w_{kS1} \cot \theta_{k,j}^{W2}(\tau) + b_k \cot \theta_{k,j}^{W2}(\tau)), \\ \theta_k^{B2}(\tau+1) &= \theta_k^{B2}(\tau) - \eta \sum_{n=1}^N (y_k^{(n)} - t_k^{(n)}) (-B2_k \tan \theta_k^{B2}(\tau)), \end{aligned} \quad (3.50)$$

$$\begin{aligned} \theta_{j,i}^{W1}(\tau+1) &= \theta_{j,i}^{W1}(\tau) - \eta \sum_{n=1}^N \sum_{k=1}^K (y_k^{(n)} - t_k^{(n)}) W2_{kj} z_j^{(n)} (1 - z_j^{(n)}) \times \\ &\quad (x_i (-W1_{ji} \tan \theta_{j,i}^{W1}(\tau)) + x_{(i+1)} W1_{j(i+1)} \cot \theta_{j,i}^{W1}(\tau) + \dots \\ &\quad + x_R W1_{jR} \cot \theta_{j,i}^{W1}(\tau) + B1_j \cot \theta_{j,i}^{W1}(\tau)), \end{aligned}$$

and

$$\theta_j^{B1}(\tau+1) = \theta_j^{B1}(\tau) - \eta \sum_{n=1}^N \sum_{k=1}^K (y_k^{(n)} - t_k^{(n)}) W2_{kj} z_j^{(n)} (1 - z_j^{(n)}) (-B1_j \tan \theta_j^{B1}(\tau)), \quad (3.51)$$

where η is the step size coefficient, τ is the time index (epoch), and z_j is the output of hidden node j .

For the purpose of maintaining a constant radial complexity, another method of performing hyperspherical backpropagation is to update the weights in Cartesian coordinates using any standard backpropagation technique, converting the new weights over to hyperspherical coordinates, resetting the magnitude parameter to the previous value, and then converting back to Cartesian coordinates. These two methods are theoretically identical (although there are limitations that must be placed on the angles since they are limited to certain regions), and there is no appreciable difference in implementation time since both

must convert the weights over to hyperspherical coordinates after each weight evaluation and update.

Here, we use hyperspherical backpropagation to find a solution weight vector at a constant radial complexity. A comparison of standard backpropagation versus hyperspherical backpropagation when classifying the hand-written OCR data set is presented in Figure 3.16. Notice that cross-validated early stopping would cause the ANN to stop training

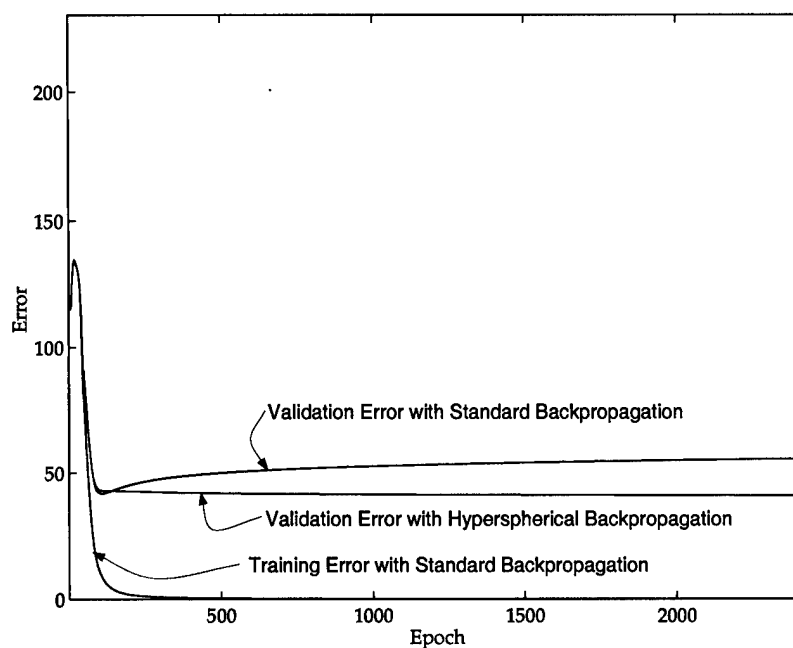


Figure 3.16 This figure demonstrates how using standard backprop to train an ANN on the hand-written character set compares to using Hyperspherical Backprop to train the ANN at a specific radial complexity. Notice that without hyperspherical backpropagation, the algorithm overtrains and begins to suffer from an increase in the validation set error.

at about 100 epochs, and without hyperspherical backpropagation to lock the radial complexity in place, the algorithm overtrains thereafter and begins to suffer from an increase in the cross-validation error which indicates less than optimal generalization. With hyperspherical backpropagation, the error on the validation set not only does not increase, it continues to decrease. This is due to the ramifications of limiting the complexity and yet continuing the training; since the corners of any two intersecting decision sigmoids can only be so sharp,

the weight updates must lower the error by shifting the global positions of the discriminant boundaries rather than going after outliers.

3.5 Summary

The determination of the initial radial complexity of an ANN based on the prior distribution used to generate our initial weight vector was discussed. This quantity determines the starting point of the ANN training algorithm, and should be of primary concern when initializing the weights. Research in the past has initialized the weights without regard for this quantity, simply using the same initial distribution for each weight regardless of the number of hidden nodes. As demonstrated, though, the initial radial complexity grows with the number of weights in the ANN so the parameters of the distribution (bounds or variance) from which each weight is drawn should change as the number of weights is changed.

By examining the behavior of the radial complexity during network training, we cast doubt on the practice of re-initializing an ANN with weights drawn from an identical distribution as previous initializations. The behavior of the radial complexity is a function of the training, the error function, and the method of training, but given these quantities, the behavior is consistent and *not* a function of the random values to which the weights are initialized (although it *is* a function of the distribution from which the weights are initialized).

Knowing the initial radial complexity and how this radial complexity behaves as the training set grows, we then showed that the radial complexity of an ANN that yields the best generalization for the full training data set can be estimated by using cross-validated early stopping on smaller size data sets, then using regression on those resultant radial complexities to obtain the radial complexity allowed by training the ANN with all available data, thereby allowing the data to constrain the decision boundaries as much as possible. With this development of radial complexity estimation, we now have a method of training an ANN with standard backpropagation techniques and yet confining the magnitude of the weight vector to a desired radial complexity so as to maintain good generalization characteristics. As an example, we showed how this method provided a better estimated general classification

accuracy than true early stopping. This technique is quite useful since the generalization characteristics of the ANN are the primary concern of the end-user.

Finally, we showed how hyperspherical backpropagation can lead to decreased validation error during training of the ANN. By limiting the radial complexity to an estimated magnitude, here estimated using cross-validational early stopping, we have forced the ANN to lower the error by shifting the location of the overall discriminant boundaries rather than overtraining on the outliers.

IV. Conclusions and Recommendations

4.1 Conclusions

Here, we reviewed that the primary consideration when training an ANN is the ability of the trained network to generalize well [7, 59]. Methods such as cross-validated early stopping and regularization attempt to find a solution at an effective complexity yielding good generalization (since the effective complexity of the network determines its generalization ability [14]). In light of the research done by Bartlett [4], the effective complexity was quantified as the magnitude of the weight vector (radius of the hypershell defined by the weight vector), and here referred to as the radial complexity.

The expected value of initial radial complexity of the ANN was shown to be an increasing function of the number of weights (based on the distribution from which each individual weight is drawn). This expected initial radial complexity is important regardless of the methods used to train the ANN since the initial radial complexity needs to be set appropriately so as to assure growth or decay into a radial complexity that yields good generalization.

The behavior of the radial complexity during training was seen to behave in a consistent manner from run to run even when the weights were re-initialized to different starting values. This behavior was shown to be independent of the data set and in fact was independent of the type of training (provided the weights were being guided toward an area of lower perceived error over the training set).

Radial complexity estimation for early stopping was shown to lead to superior generalization when used to train an ANN to a desired radial complexity, and hyperspherical backpropagation was seen to consistently decrease the validation error during training. No previous technique has attempted to obtain a solution that would remain at a specific complexity calculated to provide improved posterior probabilities as measured by the validation error.

4.2 *Recommendations for future Research*

Global minimum search techniques, questioned by Lawrence [33] for standard backpropagation, can be used with hyperspherical backpropagation with the constraint that the weight vector remains at a specific radial complexity. This hypershell global minimum (minimum obtainable training error at a given radial complexity) would yield the lowest error in that hypershell, and yet maximize the generalization capability of the ANN.

All methods used to speed up standard backpropagation, such as momentum and instantaneous backpropagation [12], are usable with radial complexity estimation and hyperspherical backpropagation, so there is immediately a plethora of algorithm tweaks to optimize training speed. Ideally, a number of solutions need to be obtained at a given radial complexity to form a committee of networks which also can help improve generalization [7]).

This research briefly mentions the relationship between the regularization coefficient, α , and the radial complexity, ρ , when using Bayesian backpropagation to train an ANN. A logical next step would be to use the technique of radial complexity estimation to determine the optimal α to use during Bayesian backpropagation to provide a result that is based on the expected generalization ability of the final weight vector.

Appendix A. Weight Update Formula

A.1 Standard Batch Backpropagation

To update each weight, we use the equation

$$w^{\tau+1} = w^{\tau} - \frac{\partial E_D(w)}{\partial w} \tau = 1, 2, \dots \quad (\text{A.1})$$

where τ is the time step, η is the step size, and w is an individual weight. Realizing that

$$E_D = \sum_{n=1}^N E_D^{(n)}, \quad (\text{A.2})$$

where, for softmax error,

$$E_D^{(n)} = - \sum_{k=1}^K t_k^{(n)} \ln(y_k^{(n)}),$$

or, for sum-squared error,

$$E_D^{(n)} = \frac{1}{2} \sum_{k=1}^K (y_k^{(n)} - t_k^{(n)})^2,$$

we can carry out the analysis as follows.

A.1.1 Second Layer Weight Update. First, we will update the second layer weights.

Using the chain rule, we see that

$$\frac{\partial E_D^{(n)}}{\partial w_{kj}} = \frac{\partial E_D^{(n)}}{\partial a_k^{(n)}} \frac{\partial a_k^{(n)}}{\partial w_{kj}}. \quad (\text{A.3})$$

From Bishop [7], we know that

$$\begin{aligned} \frac{\partial E_D^{(n)}}{\partial a_k^{(n)}} &= (y_k^{(n)} - t_k^{(n)}) \\ \frac{\partial a_k^{(n)}}{\partial w_{kj}} &= z_j^{(n)}, \end{aligned}$$

whether we use softmax or sum-squared error. Therefore

$$\frac{\partial E_D^{(n)}}{\partial w_{kj}} = (y_k^{(n)} - t_k^{(n)}) z_j^{(n)}. \quad (\text{A.4})$$

And

$$w_{kj}^{\tau+1} = w_{kj}^{\tau} - \eta \sum_{n=1}^N (y_k^{(n)} - t_k^{(n)}) z_j^{(n)}. \quad (\text{A.5})$$

A.1.2 First Layer Weight Update. The first layer weight update is carried out by setting

$$\frac{\partial E_D^{(n)}}{\partial w_{ji}} = \sum_{k=1}^K \frac{\partial E_D^{(n)}}{\partial a_k^{(n)}} \frac{\partial a_k^{(n)}}{\partial z_j^{(n)}} \frac{\partial z_j^{(n)}}{\partial a_j^{(n)}} \frac{\partial a_j^{(n)}}{\partial w_{ji}}. \quad (\text{A.6})$$

We already know $\frac{\partial E_D}{\partial a_k^{(n)}}$, so now we determine

$$\begin{aligned} \frac{\partial a_k^{(n)}}{\partial z_j^{(n)}} &= w_{kj} \\ \frac{\partial z_j^{(n)}}{\partial a_j^{(n)}} &= z_j^{(n)} (1 - z_j^{(n)}) \\ \frac{\partial a_j^{(n)}}{\partial w_{ji}} &= x_i^{(n)}. \end{aligned}$$

This leads to

$$\frac{\partial E_D}{\partial w_{ji}} = \sum_{n=1}^N \sum_{k=1}^K (y_k^{(n)} - t_k^{(n)}) w_{kj} z_j^{(n)} (1 - z_j^{(n)}) x_i^{(n)}. \quad (\text{A.7})$$

Therefore

$$w_{ji}^{\tau+1} = w_{ji}^{\tau} - \eta \sum_{n=1}^N \sum_{k=1}^K (y_k^{(n)} - t_k^{(n)}) w_{kj} z_j^{(n)} (1 - z_j^{(n)}) x_i^{(n)}. \quad (\text{A.8})$$

The step-size parameter η ($\dot{\epsilon} > 0$) has been the subject of much research and is usually chosen to speed the training process. If η is chosen to be constant for each weight update (as is sometimes the case), we note that the step size is a linearly increasing function of the training set size, N . To eliminate this dependency of the step size on N , we choose to make

$$\eta = \frac{c}{N}, \quad (\text{A.9})$$

where c is a constant.

A.2 Weight Updates with Regularization

With our regularization, we see that, in a generic case,

$$\frac{\partial S(\mathbf{w})}{\partial w} = \frac{\partial E_D(\mathbf{w})}{\partial w} + \frac{\partial E_W(\mathbf{w})}{\partial w}. \quad (\text{A.10})$$

But $\frac{\partial E_D}{\partial w}$ is the same as in Appendix A. The only change is the addition of $\frac{\partial E_W}{\partial w}$. This is simply

$$\frac{\partial E_W(\mathbf{w})}{\partial w} = \alpha w, \quad (\text{A.11})$$

which leads to

$$\begin{aligned} w_{kj}^{\tau+1} &= w_{kj}^{\tau} - \eta \left(\sum_{n=1}^N \sum_{k=1}^C (y_k^{(n)} - t_k^{(n)}) z_j^{(n)} + \alpha w_{kj}^{\tau} \right) \\ w_{ji}^{\tau+1} &= w_{ji}^{\tau} - \eta \left(\sum_{n=1}^N \sum_{k=1}^C (y_k^{(n)} - t_k^{(n)}) w_{kj} z_j^{(n)} (1 - z_j^{(n)}) x_i^{(n)} + \alpha w_{ji}^{\tau} \right). \end{aligned}$$

Appendix B. Hyperspherical Coordinate Transformation

Given a weight vector

$$\mathbf{w} = [w_1, w_2, \dots, w_W], \quad (\text{B.1})$$

we wish to generate a hyperspherical representation such that

$$\mathbf{w} = [\theta_1, \theta_2, \dots, \theta_{W-1}, \rho], \quad (\text{B.2})$$

where

$$\begin{aligned} w_1 &= \rho \cos \theta_1, \\ w_2 &= \rho \sin \theta_1 \cos \theta_2, \\ w_3 &= \rho \sin \theta_1 \sin \theta_2 \cos \theta_3, \\ &\vdots \end{aligned} \quad (\text{B.3})$$

$$\begin{aligned} w_{W-1} &= \rho \sin \theta_1 \sin \theta_2 \sin \theta_3 \dots \sin \theta_{W-2} \cos \theta_{W-1}, \\ w_W &= \rho \sin \theta_1 \sin \theta_2 \sin \theta_3 \dots \sin \theta_{W-2} \sin \theta_{W-1}. \end{aligned} \quad (\text{B.4})$$

Generating the angles is a matter of bookkeeping. All angles (except θ_{W-1}) are in the interval $[0, \pi)$, while θ_{W-1} is in the interval $[-\pi, \pi]$. The angles $\theta_1, \dots, \theta_{W-2}$ are necessary to project \mathbf{w} into the next set of dimensions while θ_{W-1} is confined to two dimensions. The radius and angles are then found as follows:

$$\begin{aligned} \rho &= \sqrt{w_1^2 + w_2^2 + \dots + w_W^2}, \\ \theta_1 &= \arccos\left[\frac{w_1}{\rho}\right], \\ \theta_2 &= \begin{cases} \arccos\left[\frac{w_2}{\rho \sin \theta_1}\right] & \theta_1 \neq 0 \\ 0 & \theta_1 = 0 \end{cases}, \end{aligned} \quad (\text{B.5})$$

$$\theta_3 = \begin{cases} \arccos\left[\frac{w_3}{\rho \sin \theta_1 \sin \theta_2}\right] & \theta_2 \neq 0 \\ 0 & \theta_2 = 0 \end{cases}, \quad (\text{B.6})$$

$$\vdots \quad (\text{B.7})$$

$$\theta_{W-2} = \begin{cases} \arccos\left[\frac{w_{W-2}}{\rho \sin \theta_1 \sin \theta_2 \dots \sin \theta_{W-3}}\right] & \theta_{W-3} \neq 0 \\ 0 & \theta_{W-3} = 0 \end{cases},$$

$$\theta_{W-1} = \begin{cases} \begin{cases} -\arccos\left[\frac{w_{W-1}}{\rho \sin \theta_1 \sin \theta_2 \dots \sin \theta_{W-2}}\right] & w_W < 0 \\ +\arccos\left[\frac{w_{W-1}}{\rho \sin \theta_1 \sin \theta_2 \dots \sin \theta_{W-2}}\right] & w_W \geq 0 \end{cases} & \theta_{W-2} \neq 0 \\ 0 & \theta_{W-2} = 0 \end{cases} \quad (\text{B.8})$$

where *arccos* denotes the principle inverse *cosine* function. With these relationships, we can freely change the hyperspherical parameters of a given weight vector to see how those changes affect the error.

Appendix C. Derivation of Angle Updates for Hyperspherical Backpropagation

For completeness, we derive the angle updates when using hyperspherical backprop.

C.1 Second Layer Angle Update

First, we will update the second layer weights. Once again, remember that

$$E_D = \sum_{n=1}^N E_D^{(n)}. \quad (\text{C.1})$$

Using the chain rule, we see that

$$\frac{\partial E_D^{(n)}}{\partial \theta_{kj}} = \frac{\partial E_D}{\partial a_k^{(n)}} \frac{\partial a_k^{(n)}}{\partial \theta_{kj}}. \quad (\text{C.2})$$

From Appendix A, we know that

$$\frac{\partial E_D}{\partial a_k^{(n)}} = (y_k^{(n)} - t_k^{(n)}). \quad (\text{C.3})$$

Now, we need to find $\frac{\partial a_k^{(n)}}{\partial \theta_{kj}}$. We know that

$$a_k^{(n)} = z_1 w_{k1} + \dots + z_j w_{kj} + z_{(j+1)} w_{k(j+1)} + \dots + z_{S1} w_{k,S1} + b_k. \quad (\text{C.4})$$

This leads to

$$\begin{aligned} \frac{\partial}{\partial \theta_{kj}} a_k^{(n)} &= \frac{\partial}{\partial \theta_{kj}} z_1 w_{k1} + \dots + \frac{\partial}{\partial \theta_{kj}} z_j w_{kj} + \frac{\partial}{\partial \theta_{kj}} z_{(j+1)} w_{k(j+1)} + \dots + \frac{\partial}{\partial \theta_{kj}} z_{S1} w_{k,S1} + \frac{\partial}{\partial \theta_{kj}} b_k \\ &= z_1 \frac{\partial}{\partial \theta_{kj}} w_{k1} + \dots + z_j \frac{\partial}{\partial \theta_{kj}} w_{kj} + z_{(j+1)} \frac{\partial}{\partial \theta_{kj}} w_{k(j+1)} + \dots + z_{S1} \frac{\partial}{\partial \theta_{kj}} w_{k,S1} + \frac{\partial}{\partial \theta_{kj}} b_k. \end{aligned}$$

Let us look at each term;

$$w_{kj} = \rho \sin \theta_1 \sin \theta_2 \sin \theta_3 \dots \sin \theta_{k(j-1)} \cos \theta_{kj}, \quad (\text{C.5})$$

so

$$\begin{aligned}
\frac{\partial}{\partial \theta_{kj}} w_{kj} &= \rho \sin \theta_1 \sin \theta_2 \sin \theta_3 \dots \sin \theta_{k(j-1)} \frac{\partial}{\partial \theta_{kj}} \cos \theta_{kj} \\
&= \rho \sin \theta_1 \sin \theta_2 \sin \theta_3 \dots \sin \theta_{k(j-1)} (-\sin \theta_{kj}) \\
&= \rho \sin \theta_1 \sin \theta_2 \sin \theta_3 \dots \sin \theta_{k(j-1)} \frac{\cos \theta_{kj}}{\cos \theta_{kj}} (-\sin \theta_{kj}) \\
&= w_{kj} \frac{(-\sin \theta_{kj})}{\cos \theta_{kj}} \\
&= -w_{kj} \tan \theta_{kj},
\end{aligned}$$

$$\begin{aligned}
\frac{\partial}{\partial \theta_{kj}} w_{k(j+1)} &= \rho \sin \theta_1 \sin \theta_2 \sin \theta_3 \dots \frac{\partial}{\partial \theta_{kj}} \sin \theta_{kj} \cos \theta_{k(j+1)} \\
&= \rho \sin \theta_1 \sin \theta_2 \sin \theta_3 \dots \cos \theta_{kj} \cos \theta_{k(j+1)} \\
&= \rho \sin \theta_1 \sin \theta_2 \sin \theta_3 \dots \frac{\sin \theta_{kj}}{\sin \theta_{kj}} \cos \theta_{k(j+1)} \cos \theta_{kj} \\
&= w_{k(j+1)} \frac{\cos \theta_{kj}}{\sin \theta_{kj}} \\
&= w_{k(j+1)} \cot \theta_{kj},
\end{aligned}$$

$$\begin{aligned}
\frac{\partial}{\partial \theta_{kj}} b_k &= \rho \sin \theta_1 \sin \theta_2 \sin \theta_3 \dots \frac{\partial}{\partial \theta_{kj}} \sin \theta_{kj} \dots \sin \theta_{kS1} \cos \theta_k \\
&= \rho \sin \theta_1 \sin \theta_2 \sin \theta_3 \dots \cos \theta_{kj} \dots \cos \theta_{kS1} \cos \theta_k \\
&= \rho \sin \theta_1 \sin \theta_2 \sin \theta_3 \dots \frac{\sin \theta_{kj}}{\sin \theta_{kj}} \cos \theta_{kS1} \cos \theta_k \\
&= b_k \frac{\cos \theta_{kj}}{\sin \theta_{kj}} \\
&= b_k \cot \theta_{kj}.
\end{aligned}$$

This leads to

$$\frac{\partial}{\partial \theta_{kj}} a_k^{(n)} = 0 + 0 + \dots + z_j (-w_{kj} \tan \theta_{kj}) + z_{(j+1)} w_{k(j+1)} \cot \theta_{kj} + \dots + z_{S1} w_{kS1} \cot \theta_{kj} + b_k \cot \theta_{kj}. \quad (C.6)$$

Therefore

$$\frac{\partial E_D^{(n)}}{\partial \theta_{kj}} = \left(y_k^{(n)} - t_k^{(n)} \right) \left(-z_j w_{kj} \tan \theta_{kj} + z_{(j+1)} w_{k(j+1)} \cot \theta_{kj} + \dots \right. \\ \left. + z_{S1} w_{kS1} \cot \theta_{kj} + b_k \cot \theta_{kj} \right),$$

and

$$\theta_{kj}^{\tau+1} = \theta_{kj}^{\tau} - \eta \sum_{n=1}^N \left(y_k^{(n)} - t_k^{(n)} \right) \left(-z_j w_{kj} \tan \theta_{kj} + z_{(j+1)} w_{k(j+1)} \cot \theta_{kj} + \dots \right. \\ \left. + z_{S1} w_{kS1} \cot \theta_{kj} + b_k \cot \theta_{kj} \right),$$

with

$$\theta_k^{\tau+1} = \theta_k^{\tau} - \eta \sum_{n=1}^N \left(y_k^{(n)} - t_k^{(n)} \right) \left(-b_{kj} \tan \theta_k \right). \quad (C.7)$$

C.2 First Layer Angle Update

Now, the first layer weight update is carried out by setting

$$\frac{\partial E_D}{\partial \theta_{ji}} = \sum_{n=1}^N \sum_{k=1}^C \frac{\partial E_D^{(n)}}{\partial a_k^{(n)}} \frac{\partial a_k^{(n)}}{\partial z_j^{(n)}} \frac{\partial z_j^{(n)}}{\partial a_j^{(n)}} \frac{\partial a_j^{(n)}}{\partial \theta_{ji}}. \quad (C.8)$$

We already know $\frac{\partial E_D^{(n)}}{\partial a_k^{(n)}}$, so now we determine

$$\frac{\partial a_k^{(n)}}{\partial z_j^{(n)}} = w_{kj} \\ \frac{\partial z_j^{(n)}}{\partial a_j^{(n)}} = z_j^{(n)} \left(1 - z_j^{(n)} \right).$$

We can find $\frac{\partial a_j^{(n)}}{\partial \theta_{ji}}$ in the same way we found $\frac{\partial a_k^{(n)}}{\partial \theta_{kj}}$

$$\frac{\partial}{\partial \theta_{ji}} a_j^{(n)} = 0 + 0 + \dots + x_i \left(-w_{ji} \tan \theta_{ji} \right) + x_{(i+1)} w_{j(i+1)} \cot \theta_{ji} + \dots + x_R w_{jR} \cot \theta_{ji} + b_j \cot \theta_{ji}. \quad (C.9)$$

This leads to

$$\begin{aligned} \frac{\partial E_D}{\partial \theta_{ji}} = \sum_{n=1}^N \sum_{k=1}^C \left(y_k^{(n)} - t_k^{(n)} \right) w_{kj} z_j^{(n)} \left(1 - z_j^{(n)} \right) & \left(x_i (-w_{ji} \tan \theta_{ji}) + x_{(i+1)} w_{j(i+1)} \cot \theta_{ji} + \dots \right. \\ & \left. + x_R w_{jR} \cot \theta_{ji} + b_j \cot \theta_{ji} \right). \end{aligned}$$

Therefore

$$\begin{aligned} \theta_{ji}^{\tau+1} = \theta_{ji}^{\tau} - \eta \sum_{n=1}^N \sum_{k=1}^C \left(y_k^{(n)} - t_k^{(n)} \right) w_{kj} z_j^{(n)} \left(1 - z_j^{(n)} \right) & \left(x_i (-w_{ji} \tan \theta_{ji}) + x_{(i+1)} w_{j(i+1)} \cot \theta_{ji} + \dots \right. \\ & \left. + x_R w_{jR} \cot \theta_{ji} + b_j \cot \theta_{ji} \right), \end{aligned}$$

and

$$\theta_j^{\tau+1} = \theta_j^{\tau} - \eta \sum_{n=1}^N \sum_{k=1}^C \left(y_k^{(n)} - t_k^{(n)} \right) w_{kj} z_j^{(n)} \left(1 - z_j^{(n)} \right) (-b_j \tan \theta_j). \quad (\text{C.10})$$

Bibliography

1. Ahmad, S. and G. Tesauro. "Scaling and Generalization in Neural Networks: A Case Study," *Proceedings of the Connectionist Models Summer School*, 2:3-10 (1988).
2. Akaho, S. "Regularization Learning of Neural Networks for Generalization," *Lecture Notes in Computer Science*, 743:99-110 (1993).
3. Angeline, Peter J., et al. "An Evolutionary Algorithm that Constructs Recurrent Neural Networks," *IEEE Transactions on Neural Networks*, 5(1):54-65 (January 1994).
4. Bartlett, Peter L. "The Sample Complexity of Pattern Classification with Neural networks: The Size of the Weights is More Important than the Size of the Network," *IEEE Transactions on Information Theory*, 44(2):525-536 (March 1998).
5. Beasley, David, et al. "A Sequential Niche Technique for Multimodal Function Optimization," *Evolutionary Computation*, 1(2):101-125 (1993).
6. Beyer, William H., editor. *CRC Standard Mathematical Tables* (27 Edition). CRC Press, Inc., 1984.
7. Bishop, Christopher M. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.
8. Bos, S. and E. S. Chng. "Using Weight Decay to Optimize the Generalization Ability of a Perceptron," *IEEE International Conference on Neural Networks*, 1:241-246 (1996).
9. Bridle, J. S. *Neurocomputing: Algorithms, Architectures and Applications*, chapter Probabilistic Interpretations of Feedforward Classification Network Outputs, with Relationships to Statistical Pattern Recognition, 227-236. Springer-Verlag, 1990. Editors: F. Fogelman Soulie and J. Hérault.
10. Buntine, Wray L. and Andreas S. Weigend. "Bayesian Back-Propagation," *Complex Systems*, 5:603-643 (1991).
11. Dalianis, P., et al. "A Study of the Generalization Capability Versus Training in Back-Propagation Neural Networks," *IEEE International Conference on Systems Man and Cybernetics*, 4:485-490 (1993).
12. Demuth, Howard and Mark Beale. *Neural Network Toolbox User's Guide*. The Math Works. For use with MATLAB.
13. Denoeux, Thierry and Régis Lengelle. "Initializing Back Propagation Networks with Prototypes," *Neural Networks*, 6:351-363 (1993).
14. Depenau, J. and M. Moller. "Aspects of Generalization and Pruning," *World Congress on Neural Networks*, 3:504-509 (1994).
15. Draye, J.P., et al. "An Inhibitory Weight Initialization Improves the Speed and Quality of Recurrent Neural Networks Learning," *Neurocomputing*, 16:207-224 (1997).

16. Duda, Richard O. and Peter E. Hart. *Pattern Classification and Scene Analysis*. John Wiley and Sons, 1973.
17. Dunne, R. A. and N. A. Campbell. "Multi-Layer Perceptrons: Robustness and the evolution of the weights through time." *Proceedings of the seventh Australian conference on neural networks*. Number 7. 155-160. 1996.
18. Fogel, David B. and Lauren C. Stayton. "On the Effectiveness of Crossover in Simulated Evolutionary Optimization," *BioSystems*, 32:171-182 (1994).
19. Hagiwara, Masafumi. "A Simple and Effective Method for Removal of Hidden Units and Weights," *Neurocomputing*, 6:207-218 (1994).
20. Hogg, Robert V. and Allen T. Craig. *Introduction to Mathematical Statistics*. Englewood Cliffs, NJ 07632: Prentice Hall, 1995.
21. Hole, Arne. "Vapnik-Chervonenkis Generalization Bounds for Real Valued Neural Networks," *Neural computation*, 8(6):1277 (1996).
22. Holland, John H. *Adaptation in Natural and Artificial Systems*. Ann Arbor: The University of Michigan Press, 1975.
23. Holland, John H. "Genetic Algorithms," *Scientific American*, 66-71 (July 1992).
24. Igel'nik, B. and Y.-H. Pao. "Estimation of Size of Hidden Layer on Basis of Bound of Generalization Error," *IEEE International Conference on Neural Networks*, 4:1923-1927 (1995).
25. Javidi, Bahram, et al. "An Optical pattern Recognition System for Validation and Security Verification," *Optical Society of America* (1994).
26. Jim, K., et al. "Effects of Noise on Convergence and Generalization in Recurrent Networks," *Advances in Neural Information Processing Systems*, 7:649-656 (1995).
27. Kinnebrock, Werner. "Accelerating the standard backpropagation method using a genetic approach," *Neurocomputing*, 6:583-588 (1994).
28. Kitano, Hiroaki. "Neurogenetic Learning: an Integrated Method of Designing and Training Neural Networks Using Genetic Algorithms.," *Physica D*, 75:225-238 (1994).
29. Kolen, John F. and Jordan B. Pollack. "Backpropagation is Sensitive to Initial Conditions," *Complex Systems*, 4:269-280 (1990).
30. Korning, P. "Training Neural Networks by Means of Genetic Algorithms Working on Very Long Chromosomes," *International journal of neural systems*, 6(3):299 (1995).
31. Lange, R. and R. Maenner. "Quantifying a Critical Training Set Size for Generalization and Overfitting Using Teacher Neural Networks," *ICANN -Proceedings*, 1:495-500 (1994).
32. Larsen, J. and L. K. Hansen. "Generalization Performance of Regularized Neural Network Models," *Neural Networks for Signal Processing*, 4:42-51 (1994).

33. Lawrence, S., et al. "Local Minima and Generalization," *IEEE International Conference on Neural Networks*, 1:371-376 (1996).
34. Lee, K.W. and H.N. Lam. "Optimizing Neural Network Weights using Genetic Algorithms: A Case Study," *IEEE*, 1384-1388 (1995).
35. Lee, Soo-Young and Minho Lee. "Curvature Smoothing and Improved Generalization by Hybrid Back-Propagation/Hebbian Learning Rule," *World Congress on Neural Networks*, 1:596-599 (1995).
36. Leonard, J. and M.A. Kramer. "Improvement of the Backpropagation Algorithm for Training Neural Networks," *Computers chem. Eng.*, 14(3):337-341 (1990).
37. Lin, Chin-Teng and others. "Fuzzy Adaptive Learning Control Network with On-Line Neural Learning," *Fuzzy Sets and Systems*, 71:25-45 (1995).
38. MacKay, David J. *Bayesian Methods for Adaptive Models*. PhD dissertation, California Institute of Technology, 1991.
39. MacKay, David J. "Bayesian Interpolation," *Neural Computation*, 4:415-447 (1992).
40. MacKay, David J. "The Evidence Framework Applied to Classification Networks," *Neural Computation*, 4:720-736 (1992).
41. MacKay, David J. "A practical Bayesian Gramework for Backpropagation Networks," *Neural Computation*, 4:448-472 (1992).
42. Mattrea, D. and F. Palmieri. "New Bounds for Correct Generalization," *IEEE International Conference on Neural Networks*, 2:1051-1055 (1997).
43. Michalewicz, Zbigniew. *Genetic Algorithms + Data Structures = Evolution Programs* (3 Edition). Springer, 1996.
44. Monasson, Remi and Riccardo Zecchina. "Weight Space Structure and Internal Representations: A Direct Approach to Learning and Generalization in Multilayer Neural Networks," *Physical Review Letters*, 76(12):2205 (1996).
45. Neal, Radford M. *Bayesian Training of Backpropagation Networks by the Hybrid Monte Carlo Method*. Technical Report, Dept. of Computer Science, University of Toronto, 1992. Technical Report CRG-TR-92-1.
46. Neal, Radford M. *Bayesian Learning for Neural Networks*. Springer, 1996.
47. Negoita, M. Gh. and Dan Mihaila. "Intelligent Techniques Based on Genetic Evolution with Applications to Neural Networks Weights Optimization," *Proceedings of the International Congress on Cybernetics*, 14:955-962 (1995).
48. Nguyen, Derrick and Bernard Widrow. "Improving the Learning Speed of 2-Layer Neural Networks by Choosing Initial Values of the Adaptive Weights," *IEEE International Joint Conference of Neural networks*, 3:21-26 (1990).
49. Oh, J. H., et al. "Generalization in Two-Layer Neural Networks," *Progress in Neural Processing*, (1):18-31 (1995).

50. Oh, Jong-Hoon and Kukjin Kang. "Generalization in Two-Layer Neural Networks," *Progress in Neural Processing*, 1:18-31 (1995).
51. Omlin, C. W. and C. L. Giles. "Pruning Recurrent Neural Networks for Improved Generalization Performance," *Neural Networks for Signal Processing*, 4:690 (1994).
52. Onoda, T. "Experimental Analysis of Generalization Capability based on Information Criteria," *IEEE International Conference on Neural Networks*, 1:114-119 (1996).
53. Pan, Zhengjun and Lishan Kang. "Evolving Both the Topology and Weights of Neural Networks," *Parallel Algorithms and Applications*, 9:299-307 (1996).
54. Park, Eui H. and others. "Adaptive Learning of Human Motion by a Tlerobont Using a Neural Network Model as a Teacher," *Computers and Industrial engineering*, 27(1-4):453-456 (1994).
55. Poggio, Tomaso and Federico Girosi. "Networks for Approximation and Learning," *Proceedings of the IEEE*, 78(9):1481-1497 (September 1990).
56. Poggio, Tomaso and Federico Girosi. "Regularization Algorithms for Learning That are Equivalent to Multilayer Networks," *Science*, 247:978-982 (1990).
57. Porto, Vincent W., et al. "Alternative Neural Network Training Methods," *IEEE Expert*, 16 (June 1995).
58. Ramamurti, V. and J. Ghosh. "Improved Generalization in Localized Mixture of Experts Networks," *Intelligent Engineering Systems Through Artificial Neural Networks*, 7:5-10 (1997).
59. Ripley, B. D. *Pattern Recognition and Neural Networks*. Cambridge, 1996.
60. Rogers, Steven K. and Matthew Kabrisky. *An Introduction to Biological and Artificial Neural Networks for Pattern Recognition, TT 4*. SPIE Optical Engineering Press, 1991.
61. Ruck, Dennis W., et al. "The Multilayer Perceptron as an Approximation to a Bayes Optimal Discriminant Function," *IEEE Transactions on Neural Networks*, 1(4):296-298 (December 1990).
62. Rudolph, S. "On Topology, Size and Generalization of Non-Linear Feedforward Neural Networks," *Neurocomputing*, 16(1):1-22 (1997).
63. Sarkar, D. "Empirical Estimation of Generalization Ability of Neural Networks," *Proceedings- SPIE The International Society for Optical Engineering*, (2760):54-60 (1996).
64. Saseetharran, M(Sasheel). "Experiments That Reveal the Limitations of the Small Initial Weights and the Importance of the Modified Neural Model," *IEEE* (1996).
65. Schmidt, M. "A Unification of Genetic Algorithms, Neural Networks and Fuzzy Logic: The GANNFL Approach," *Artificial Neural Networks - ICANN 96.*, (1112):495 (1996).
66. Schraudolph, Nocol N. and Terrence J. Sejnowski. "Tempering Backpropagation Networks: Not All Weights are Created Equal," *Advances in Neural Information Processing Systems*, 8:563-569 (1996).

67. Sexton, Randall S., et al. "Toward Global Optimization of Neural Networks: A Comparison of Genetic Algorithm and Backpropagation," *Decision Support Systems*, 22:171-185 (1998).
68. Sternieri, Armando and Paolo Anelli, "Evolution Programs to Learn Weights and Topology of Neural Networks." Internet Citation, September 1996. <http://www.fis.unipr.it/~anelli/articoli/AIIAga/rivista/rivista.html>.
69. Tang, Chuan Zhang and Hon Keung Kwan. "Parameter effects on convergence speed and generalization capability of backpropagation algorithm," *Int. J. Electronics*, 74(1):35-46 (1993).
70. Thomas, John B. *Introduction to Probability*. Springer-Verlag, 1986.
71. Trafalis, Theodore B. and Tarek A. Tutunji. "A Quasi-Newton Barrier Function Algorithm for Artificial Neural Network Training with Bounded Weights," *ASME Press Series on International Advances in Design Productivity*, 4:161-166 (1994).
72. Tsukuda, Y., et al. "Investigation of Generalization Ability by Using Noise to Enhance MLP Performance," *IEEE International Conference on Neural Networks*, 5:2795-2798 (1995).
73. Turmon, M. J. and T. L. Fine. "Assessing Generalization of Feedforward Neural Networks," *IEEE International Symposium on Information Theory*, 168 (September 1995).
74. Turmon, M. J. and T. L. Fine. "Empirically Estimating Generalization Ability of Feedforward Neural Networks," *World Congress on Neural Networks*, 1:600-605 (1995).
75. Tvetter, Don. "Getting a Fast Break with BACKPROP," *AI Expert*, 6:36-44 (1991).
76. Tzafestas, S.G., et al. "On the overtraining phenomenon of backpropagation neural networks," *Mathematics and Computers in Simulation*, 40:507-521 (1996).
77. Vapnik, V. N. and A. YA. Chervonenkis. "On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities," *Theory of Probability and its Applications*, 16(2):264-280 (1971).
78. Vogl, T. P., et al. "Accelerating the Convergence of the Back-Propagation Method," *Biological Cybernetics*, 59:257-263 (1988).
79. Vrahatis, M.N., et al. "On the Acceleration of the Backpropagation Training Method," *Nonlinear Analysis, Theory, methods and Applications*, 30(7):4551-4554 (1997).
80. Waibel, Alexander, et al. "Phoneme Recognition Using Time-Delay Neural Networks," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37(3):328 (March 1989).
81. White, David and Panos Ligomenides. "GANNet: A Genetic Algorithm for Optimizing Topology and Weights in Neural Network Design," *Lecture notes in computer science*, (686):322-327 (1993).
82. White, Halbert. "Learning in Artificial Neural Networks: A Statistical Perspective," *Neural Computation*, 1:425-464 (1989).

83. White, Halbert. "Connectionist Nonparametric Regression: Multilayer Feedforward Networks Can Learn Arbitrary Mappings," *Neural Networks*, 3:535-549 (1990).
84. Wilson, D. Randall and Tony R. Martinez. "Instance-Based Learning with Genetically Derived Attribute Weights," *Proceedings of the IASTED International Conference Artificial Intelligence, Expert Systems, and Neural Networks*, 19-21 (August 1996).
85. Yoon, H. and J. H. Oh. "Learning and Generalization in Higher-Order Perceptrons," *Philosophical Magazine B*, 77(5):1557-1564 (1998).
86. Zheng, Baoyu, et al. "Digital Mammography: Mixed Feature Neural Network with Spectral Entropy Decision for Detection of Microcalcifications," *IEEE Transactions on Medical Imaging*, 15(5):589 (October 1996).

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE September 1998		3. REPORT TYPE AND DATES COVERED PhD Dissertation
4. TITLE AND SUBTITLE Radial Complexity Estimation for Improved Generalization in Artificial Neural Networks			5. FUNDING NUMBERS	
6. AUTHOR(S) Lemuel R. Myers, Jr, Capt, USAF				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology, WPAFB OH 45433-6583			8. PERFORMING ORGANIZATION REPORT NUMBER AFIT/DS/ENG/98-14	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Capt Thomas Rathbun AFRL/SNAS, Area B, Bldg 23 Wright-Patterson AFB, OH 45433 255-6329 x2623			10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES				
12a. DISTRIBUTION AVAILABILITY STATEMENT Approved for Public Release; Distribution Unlimited			12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words) When training an artificial neural network (ANN) for classification using backpropagation of error, the weights are usually updated by minimizing the error on the training set. As training ensues, overtraining may be observed as the network begins to memorize the training data because, as the magnitude of the weight vector grows, the decision boundaries become overly complex. It is then important to initialize the weights with consideration to the importance of the weight vector magnitude. The expected value of the magnitude of the initial weight vector is here derived for the separate cases of each weight drawn from a normal or uniform distribution. The usefulness of this derivation is universal since the magnitude of the weight vector plays such an important role in the formation of the classification boundaries. One way to overcome the overtraining problem is to stop the training early, which limits the magnitude of the weight vector below what it would be if the training were allowed to continue until a near-global training error minimum were found. Here, the relationship between training data set size and the radial complexity providing good generalization results is empirically established using cross-validational analysis on small subsets of the training data. These results are then used to estimate at what weight vector magnitude the training should be stopped when using the full data set. The general classification ability of an ANN trained in this manner is shown to increase the percentage of correctly classified test data points by an average of 1.5% over that of one trained using true cross-validational early stopping on a smaller data set. The technique of hyperspherical backpropagation is also introduced and shown to be useful in lowering the validation error during training.				
14. SUBJECT TERMS Pattern Classification, Artificial Neural Networks, Multi-Layer Perceptrons, Regularization, Discriminant Boundaries, Radial Complexity			15. NUMBER OF PAGES 97	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL	